VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra měřicí a řídicí techniky

Analýza a detekce definovaných geometrických prvků z obrazového signálu

Video Signal Analysis and Detection of Geometrical Elements

2011 Marcel Nečesaný

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra měřicí a řídicí techniky

Zadání bakalářské práce

Student: **Marcel Nečesaný**

Studijní program: B2649 Elektrotechnika

Studijní obor: 2601R004 Měřicí a řídicí technika

Téma: Analýza a detekce definovaných geometrických prvků z obrazového signálu
Video Signal Analysis and Detection of Geometrical Elements

Zásady pro vypracování:

1. Rozbor problematiky měření a zpracování obrazu.
2. Návrh a realizace metod pro snímání a zpracování obrazu.
3. Řešení systému pro analýzu obrazu a detekce definovaných geometrických prvků.
4. Srovnání naměřených a analyzovaných dat s teoretickými předpoklady.
5. Zhodnocení dosažených výsledků.

Seznam doporučené odborné literatury:

1. VÍT, V. - KUBA, P. *Televizní technika: studiové zpracování televizního signálu.* 1. vyd. Praha: BEN-technická literatura, 2000. 208 s. ISBN 80-86056-88-0.
2. VÍT, V. *Televizní technika: přenosové barevné soustavy* . 1. vyd. Praha: BEN-technická literatura, 1997. 720 s. ISBN 80-86056-04-X.
3. VÍT, V. - GREGORA, P. *Televizní technika: Zařízení pro přenos a vysílání televizního signálu.* 1. vyd. Praha: BEN-technická literatura, 2000. 176 s. ISBN 80-86056-89-9.
4. BURKHARD, K. *USB: měření, řízení a regulace pomocí sběrnice USB.* [CD-ROM. PC & elektronika] 1. vyd. Praha: BEN-technická literatura, 2002. 256 s. ISBN 80-7300-073-3.
5. BEZDĚK, M. *Elektronika III.* 1. vyd. České Budějovice: Kopp, 2004. 236 s. ISBN 80-7232-241-9.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Zdeněk Macháček, Ph.D.**

Datum zadání: 19.11.2010
Datum odevzdání: 06.05.2011

VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA * FAKULTA elektrotechniky a informatiky * OSTRAVA *

doc. Ing. Jiří Koziorek, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě ……. Podpis studenta

Prohlašuji, že

jsem byl seznámen s tím, že na moji bakalářskou/diplomovou práci se plně vztahuje zákon č 121/2000 Sb. – autorský zákon, zejména §35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a §60 – školní dílo.

beru na vědomí, že Vysoká škola báňská – technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně ke své vnitřní potřebě bakalářskou/diplomovou práci užít (§35 ods. 3).

souhlasím s tím, že jeden výtisk bakalářské/diplomové práce bude uložen v Ústřední knihovně VŠB-TUO k prezenčnímu nahlédnutí a údaje o bakalářské/diplomové práci budou zveřejněny v informačním systému VŠB-TUO.

beru na vědomí, že odevzdáním své práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, bez ohledu na výsledek její obhajoby.

V Ostravě ……. Podpis studenta

Poděkování

Rád bych tímto poděkoval vedoucímu bakalářské práce Ing. Zdeňku Macháčkovi, Ph.D. za poskytnutí teoretických podkladů a praktických rad. Dále bych chtěl poděkovat mému kamarádovi Aleši Kurečkovi za pomoc při řešení technických problémů.

Abstrakt:

Tato práce se zabývá návrhem a realizací systému pro analýzu a detekci geometrických prvků z obrazového signálu. Hlavní úkol je detekce předdefinovaných geometrických prvků. Systém se skládá z kamery snímající obraz a vývojového kitu, který obraz zpracovává a posílá ho přes sériové rozhraní do počítače. Navrhnuté a implementované algoritmy a metody analýzy použité v této práci je možno použít pro detekci jakýchkoliv (předdefinovaných) geometrických prvků. Každá metoda byla vytvořena a otestována v programovacím jazyce C#. Následně byl vybrán nejefektivnější způsob detekce geometrických prvků. Detekci geometrických prvků ještě předchází potřebné úpravy signálu. Tyto úpravy jsou odstranění šumu, binarizace obrazu, skeletonizace obrazu. Poté je obraz segmentován, to znamená, že je oddělen na jednotlivé znaky. Tyto znaky jsou převedeny na vertexy a následně pospojovány úsečkami. Nakonec je provedena vektorizace. Výsledek těchto metod a také dílčích úprav obrazu bude zobrazen v zobrazovacím prvku objektového programovacího jazyka.

Klíčová slova:

Kamera, Detekce geometrických prvků, Analýza obrazu, Odstranění šumu, Binarizace, Skeletonizace, Segmentace, Vertex

This work deals with design and implementation of software for video signal analysis and detection of geometrical elements. The main task is to detect known geometrical elements (for example numbers). The system consists of camera which scans the picture and the development kit, which processes the image and sends it via serial port on your computer. Designed and implemented algorithms and methods of analysis used in this study can be used to detect any (predefined) of geometrical elements. Each method has been created ant tested in programming language C#. Subsequently, it was selected the most efficient and reliable detection algorithm for detection of geometrical elements. Before detection of geometric elements we need to make some adjustments of the signal. These adjustments are noise removing, binarization of the picture, skeletonization.

After the image is segmented that means is separated into individual characters. These characters are converted to vertices connected by lines. Finally tracing is made. The outcome of these methods as well as incremental adjustments to an image will appear in the display element of object programming language.

Key words:

Camera, Detection of geometrical elements, Image analysis, Removing of noise, Binarization, Skeletonization, Segmenting, Vertice

Seznam použitých symbolů a zkratek:

CIF	Common Intermediate Format, rozlišení videa 352 × 288 v pixel
CMOS	Comlpementary Metal Oxid Semicondictor
CMUcam3	Kamera vývojového kitu
FAT16	File Allocation Table, souborový systém, verze FAT16
GNU	Volně šiřitelný software, inspirovaný operačním systémem unixového stylu
GPIO	General Purpose Input/Output, je vstupně výstupní rozhraní zařízení
ISP	In Systém Programming, programování bez nutnosti vyjmutí
I2C	Inter-Integrated Circuit, počítačová sériová sběrnice
JPEG	Joint Photographic Experts Group, metoda ztrátové komprese obrázků
MOSI	Master Output Slave Input
MMC	MultiMediaCard, standard paměťové karty s technologií paměti flash
PC	Personal computer, osobní počítač
QCIF	Quarter Common Intermediate Format, rozlišení videa 176 × 143 v pixel
RGB	Barevný model, červená – zelená – modrá
Software	Programové vybavení, vykonávající nějakou činnost
TTL	Transistor – transistor logic, tranzistorově-tranzistorová logika
UART	Synchroní a asynchroní seriové rozhranní
USB	Universal Serial Bus, univerzální seriová sběrnice

Obsah

1. Úvod

Cílem bakalářské práce je navrhnout a realizovat systém pro analýzu a detekci geometrických prvků z obrazového signálu. Systém se skládá ze samotné kamery, vývojového kitu, který obraz zpracovává a posílá přes sériové rozhraní do počítače. Sledovaným předmětem je stránka s geometrickými prvky a číslicemi.

V dnešní době se požívá mnoho druhů kamer. Využívají se jak černobíle, tak i barevné verze těchto kamer. Vývojový kit použitý, jako součást vestavěného řídicího systému, využívá kameru vybavenou barevným rozlišení. Maximální možné rozlišení kamery je 352x288 pixel.

Pomocí kamery je pořízen snímek a následně se spustí algoritmus pro detekci, požadovaných geometrických prvků. Obraz je filtrován, binarizován následně je ztenčen na šířku jednoho pixelu (skeletonizace). Obraz je dále segmentován to znamená, že je rozdělen na jednotlivé znaky. Následovně jsou vytvořeny referenční seznamy se znaky, které jsou detekovány. Tyto referenční seznamy jsou porovnávány s jednotlivými prvky v obrazovém signálu. Pokud se prvek z referenčního seznamu podobá prvku v obrazu tak se detekovaný prvek překreslí prvkem z referenčního seznamu.

Výsledný obraz je zobrazován v zobrazovacím prvku, který je realizován v jazyce C#.

V následující kapitole je popsáno seznámení se s kamerou a popis barevného modelu, se kterým kamera pracuje. Podrobně rozebrán princip a funkčnost kamery.

Seznámení se s typy přenosů signálu je v další kapitole. Jsou podrobně popsány druhy přenosů signálu a zařízení potřebné pro jejich přenos.

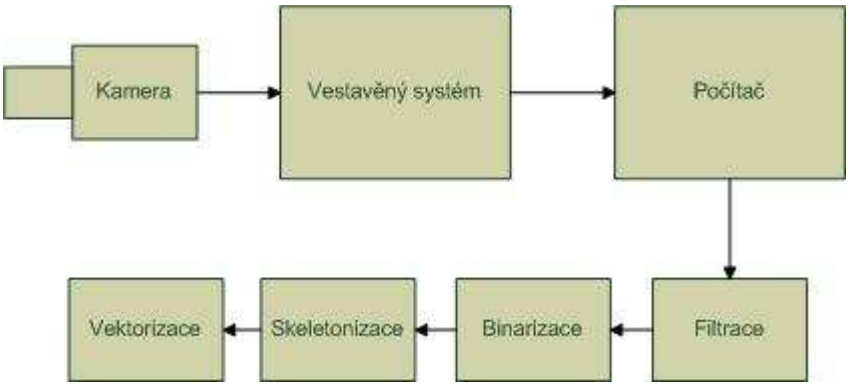
V další kapitole je rozebrána detekce geometrických prvků, kterou provádíme pomocí barevných odstínů. Na příkladě je vidět, že aplikace napsaná v matematickém prostředí Matlab je schopná se slušnou přesností detekovat geometrické prvky pomocí jejich barevných odstínů (podpora pouze pro RGB).

Problematika úpravy signálu pro detekování geometrických prvků je podrobně rozebrána v následující kapitole. Konkrétně se jedná o binarizaci (převod obrázku s RGB na černobílý) ,filtraci za pomocí Gaussova filtru, ztenčení obrazu pomocí metody Skeletonizace.

Obraz je ztenčen na šířku jednoho pixelu pro snadnější vektorizaci.

Princip segmentace a převodu na vertexy je popsán v další kapitole. Obraz je oddělen na jednotlivé znaky. Tyto znaky jsou následně uloženy do seznamů. Dále jsou vytvořeny jednotlivé referenční seznamy s prvky, které jsou detekovány. A tyto seznamy se porovnávají s prvky v obrazovém signálu. Pokud je referenční prvek stejný jako prvek v obrazovém signálu tak ho referenční prvek překreslí.

Kapitola věnující se testování celého aplikace popisuje testování a návaznost na jednotlivých částech systému. Těmi jsou algoritmus úpravy signálu a samotná detekce geometrických prvků. Následně je popsáno uživatelské rozhraní aplikace.



Obr. 1. Blokové schéma systému pro analýzu a detekci geometrických tvarů z obrazového signálu.

Sledované geometrické prvky jsou zaznamenány kamerou. Obrazový signál je uložen do vestavěného systému, který je tvořen vývojovým kitem. Tento systém zpracovává obrazový signál a posílá ho do počítače přes sériové rozhraní. V počítači jsou pro detekci geometrických prvků na tento obrazový signál aplikovány následující metody. V první řadě je to metoda filtrace realizována pomocí Gaussovského rozmazání. Dále je obrazový signál binarizován tzn. obraz je zaostřen a převeden na černobílý. Následuje metoda skeletonizace, která nám veškeré prvky v obrazovém signálu ztenčí na šířku jednoho pixelu. A v poslední řadě je obraz vektorizován. Metodě vektorizace předchází ještě segmentace. Segmentací jsou prvky v obrazu odděleny. Segmentované prvky jsou převedeny na vertexy a pospojovány úsečkami. Následně jsou vytvořeny referenční seznamy s jednotlivými detekovanými prvky. Pokud jsou detekované prvky podobné těm v referenčních seznamech. Tak je prvky z referenčních seznamů překreslí. To znamená, že prvek je vektorizován. Výsledek aplikování těchto metod je zobrazen pomocí zobrazovacího prvku, který zprostředkovává programovací jazyk C#.

2. Zpracování a přenos obrazového signálu z kamery

2.1 Kamerové zařízení

Kamera je součástí vestavěného řídicího systému pro analýzu obrazu. CMUcam3 je kamerové zařízení určené zejména pro robotiku a výzkum. Skládá se z malé videokamery a 32bit mikrokontroléru LPC2106 od NXP (Philips), se sériovým rozhraním. CMUcam má také velmi malé rozměry. Z těchto důvodů je poměrně využívaná pro výrobu malých mobilních robotů, pozorování, senzorové sítě, interaktivní hračky, rozpoznávání objektů a sledování.

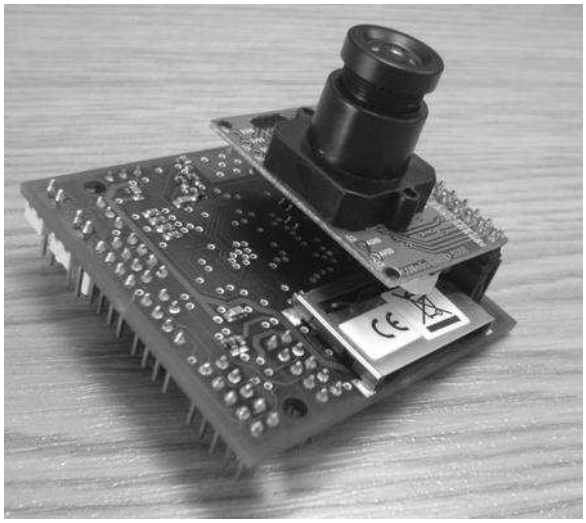
C3088 je 1/4 " barevná kamera modulovaná s digitálním výstupem. Používá OmniVision CMOS obrazový snímač OV6620. Kombinací technologie CMOS, spolu se snadno použitelným digitálním rozhraním. Kamera umožňuje nízkonákladové řešení pro vyšší kvalitu obrazu a video aplikací.

Obr. 2. Kamera [8]

Digitální video port dodává kontinuální 8/16 bit-široký obraz datového toku. Všechny funkce fotoaparátu, jako je expozice, gama, zisk, vyvážení bílé, barevné matice, jsou programovatelné přes rozhraní I2C. Tato kamera je určená pro programování zejména v operačních systémech Windows, tak i v operačních systémech Linux. Vlastní C kód, může být vyvinut pro použití GNU nástroje spolu se souborem open source knihoven. Jedná se o svobodný software unixového typu a také o volně přístupné knihovny pro dané zařízení. Spustitelné soubory mohou být nahrány pomocí sériového portu RS232.

Vstupní napájecí napětí je přivedeno na desku přes 5 voltový regulátor. Ideální napájecí napětí pro desku je mezi 6 a 15 DCV. Zdroj také musí dodávat minimálně 150 mA. Případně připojené servo pohony na portu, mohou být napájená přímo z desky nebo z externího zdroje přehozením propojky na desce. Pokud je vyžadován větší proud, než je maximálně povolený, dojde k resetu procesoru. Běh více servo pohonů nebude úspěšný. Porty, určené především pro servo pohony, lze použít také jako výstupy pro všeobecné digitální použití. [7]

Vývojový kit je složen z modulu se seriovým rozhraním, nebo s rozhraním USB 2.0. Modul umožňuje nahrát video data získané z kamery do počítače. Elektronika desky je speciálně určena pro vyhodnocení digitálního obrazu pomocí OmniVision obrazového snímače. Vyhodnocovací senzor je do 5Mega Pixel. Rychlost přenosu je 48Mbps. Modul podporuje operační systém Windows a operační systém Linux. Další výhodou modulu s rozhraním USB je, že není vyžadováno externí napájení. Modul se seriovým rozhraním musí být externě napájen. [7]



Obr. 3. Vývojový kit [8]

Kamera je založena na technologii CMOS. Jde o technologii používanou ve většině integrovaných obvodů. Metal-oxid-semiconductor, odkazuje na fyzickou strukturu tranzistorů. Obsahuje kovovou řídící elektrodu. Kamera integruje obraz pole, zpracování signálu, načasování a ovládání obvodů, vše na jednom čipu. Použití jednoduchého obvodu nabízí jedinečné výhody, jako je nízká spotřeba, malé rozměry spolu s odpovídající kvalitou obrazu.

Kamera C-CamA a C-Cam2A mají stejné vlastnosti, liší se pouze jiným tvarem. Snímá černo-bíle, formát objektivu je 1/4", obrazové čidlo OV5116, objektiv f4.9mm, F2.8.

Kamera C-Cam8A je barevná, formát objektivu má 1/4". Používá obrazové čidlo OV7949, nebo OV6620. Objektiv f6.0mm, F1.8. [7]

Obr. 4. Různé druhy optik [8]

2.2 Fyzické propojení s PC

Vyvojový kit je propojen s počítačem pomocí sériového portu. Stejně jako TTL pro komunikaci s mikrokontrolérem. Vývojový kit využívá z celého sériového portu jen 3 piny z 10 pinů. Je v konfiguraci 2x5 pin. Odpovídá standardní 9-ti pinu plochého kabelu, sériové zásuvky a 10-ti pin sériové hlavičky. Sériová propojka musí být na svém místě, například během módu importování kódu do procesoru. TTL výstupní piny jsou mezi 0 a 3.3V, ale toleruje se i 5V.

Vyvojový kit dále obsahuje digitální vstupy/výstupy. Ty jsou vlastně vstupně výstupní digitální rozhraní zařízení kamery. Umožňuje přístup ke druhé UART. UART je synchronní a asynchronní sériové rozhranní. Dále obsahuje různě řízené napájení na pinech a ISP pin.

Povolené napájení - Pokud je nízké napájecí napětí, hlavní regulátor zařízení způsobující povolení čerpání proudu od hodnoty 0.01uA, je vypnut. Veškeré zařízení je vypnuto a ztraceny veškeré informace. Pokud je odběr větší nastane opětovné spuštění. Deska se restartuje. Pin je ve výchozím nastavení.

AUX napájení – Tento pin může být nastaven pro napájení přímo z desky nebo z externího zdroje. Pin je standardně napájen 3.3 V. Při přesunutí rezistoru R11 a přidání propojky rezistoru na místo R6, pin bude připojen na 5 V napájecí napětí, před 5-ti voltový regulátor.

CAM RESET - Tento pin lze použít jako externí I / O, pokud stav kamery není nutné znát. Standardně tento pin resetuje modul kamery a neměl by být používán.

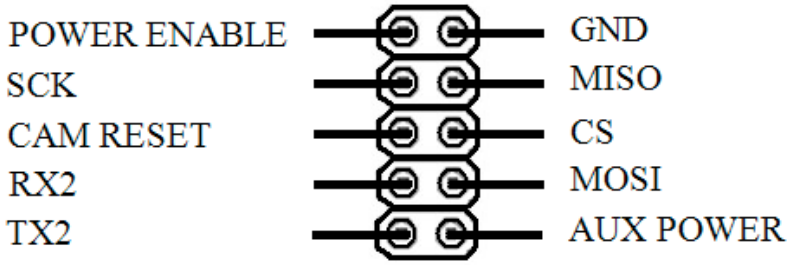
TX2 a RX2 – vysílací a přijímací pin na UART2 není na úrovni posunu, proto nemůže být přímo připojen k PC, neboť nemá TTL externí zařízení. Tento pin lze použít také jako GPIO.

CS – Slouží na znovuobnovení, jestli pin drží stav nízké úrovně, LPC2106 spustí zaváděcí režimu. Reboot může být externě vyvolaný pulzující energie umožněný pinem. Standardně je tento pin řízen MMC. Tento pin lze použít také jako GPIO nebo jako čip SPI, když MMC karta není vložena.

MOSI a MISO - Obvykle jsou tyto piny ovládány MMC. Lze je použít jako GPIO, nebo jako výstup SPI, pokud MMC karta není vložena.

SCK - Obvykle tento pin je ovládán řadičem MMC. Tento pin lze použít také jako GPIO nebo jako SPI pin hodin, když MMC karta není vložena.

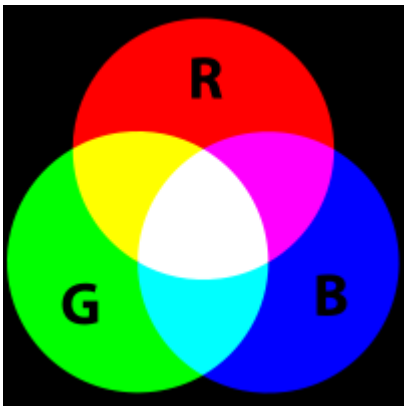
[7]



Obr. 5. GPIO port [6]

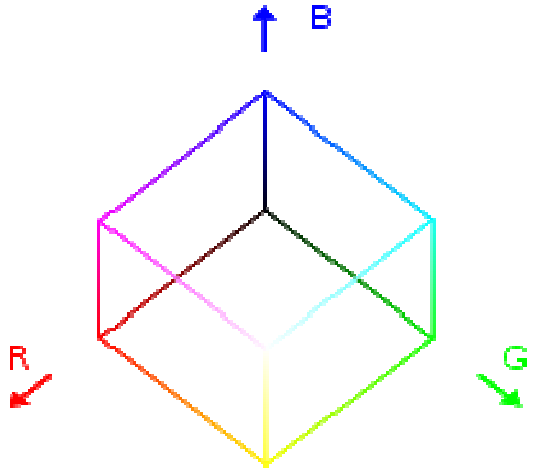
2.3 Princip zpracování barevného obrazu

Promítání základních barev světla na obrazovku ukazuje aditivní barvy, kdy se dvě překrývají. Kombinace všech tří – červené, zelené a modré v odpovídající intenzitě vytváří bílou barvu. RGB model je aditivní barevný model, ve kterém je smícháno společně červené, zelené a modré světlo různými cestami k reprodukci obsáhlého pole barev. Hlavní účel modelu RGB je snímání a zobrazování obrázků (obrazu) v elektronických systémech.



Obr. 6. Aditivní míchání barev [6]

Název modelu je odvozen z počátečních písmen základních barev. Model sám o sobě nedefinuje, co je přesně myšleno červenou, zelenou a modrou barvou, proto výsledek smíchání složek není přesný, je relativní. Model je možné také zobrazit jako krychli. Každá z kolmých hran udává velikost intenzity barevných složek. Kterýkoli bod se souřadnicemi R, G, B. Představuje hodnotu výsledné barvy.



Obr. 7. Barevný model RGB [6]

Číselná zobrazení základních barev, které mají vlnové délky 630, 530 a 450nm. Velikost se udává dekadicky (procentuálně), nebo jako určitý počet bitů (barevná hloubka). Pro 8 bitů je rozsah 0 – 255, pro 16 bitů je rozsah 0 – 65535. Čím je velikost větší, tím s vyšší intenzitou se barva zobrazuje, je světlejší. Tento jev je znázorněn v barevném RGB modelu. Hodnoty v barevném modelu se zvětšují podél osy x (červená), y (zelená) a z (modrá). Pokud jsou veškeré barvy na minimální hodnotě, výsledkem je černá Naopak při maximální hodnotě všech barev, výsledkem je bílá.

Jednotlivé barvy jsou základní barvy v rozsahu 0 (minimum) a 1 (maximum). Mnoho rovnic používají následujících rozsahů. Plná intenzita červené barvy je 1, 0, 0. Hodnoty mohou být zapsány také procentuálně, 0% (minimum) 100% (maximum). Pro červenou to je 100% 0% 0%. Číselně v rozsahu od 0 – 255, to znamená pro červenou barvu hodnoty 255, 0, 0. Hexadecimální podobně pro tutéž barvu vychází FF, 00, 00.

R	G	B	BARVA
0	0	0	černá
255	0	0	červená
0	255	0	zelená
0	0	255	modrá
255	255	0	žlutá
255	0	255	purpurová
0	255	255	azurová
255	255	255	bílá

Tab. 1. 8 bit barevné spektrum v detailu [6]

Běžné použití tohoto RGB barevného modelu je rozdělení barevného elektronového paprsku, LCD displeje, plazmového displeje. Například monitor, nebo televize. Každý pixel na obrazovce může být

reprezentován jako hodnoty pro červenou, zelenou a modrou. Hodnoty jsou převedeny do intenzity, nebo elektrického napětí přes gama korekci. Myšlená intenzita je reprodukována na displej. Běžné displeje využívají na 1 pixel 24 bitů, při 8 bitovém rozvržení. Každý pro červenou, zelenou a modrou barvu. Tento systém rozvržení dosahuje kombinace 16 777 216 (256^3 nebo 2^{24}).

[6]

3. Návrh a implementace metod pro rozpoznání geometrických prvků v obrazu

Při návrhu následujících metod se vychází jak s tvarů jednotlivých geometrických prvků, tak i z jejich barev. Jsou navrženy a implementovány dvě hlavní metody. První metoda se nezaobírá přímo tvary, ale barvami jednotlivých tvarů a podle toho je také detekuje. Druhá metoda napřed obraz upraví a až pak detekuje jednotlivé tvary. Nevšímá si barev, ale tvarů, detekce je provedena vektorově.

Tyto dvě hlavní metody jsou:

1. Detekce geometrických prvků pomocí barev.
2. Vektorová detekce geometrických prvků.

Vektorová detekce geometrických prvků není tak jednoduchá jako detekce podle barev. Musí ji předcházet další metody pro úpravu obrazu.

Tyto metody jsou:

1. Filtrace pomocí Gaussovské funkce.
2. Binarizace obrazu.
3. Skeletonizace obrazu.
4. Segmentace a převod na vertexy.

3.1 Rozbor detekce geometrických prvků pomocí barev

Jelikož jsou detekovány geometrické prvky pomocí barev, tak se v prvé řadě vypočte rozdíl mezi zvolenou barvou a barvou v matici.

Tento rozdíl se vypočítá jako Euklidovská metrika (vzdálenost) mezi vektory barvy obrazu a zvolené barvy řeší tento vztah:

$$\Delta c = \sqrt{(v_{11} - v_{21})^2 + (v_{12} - v_{22})^2 + (v_{13} - v_{23})} \quad (1)$$

Tento rozdíl je uložen do matice diference. Pro přesnější určení diference i u obrázků s malým kontrastem je tato matice ještě upravena na hodnoty z intervalu <0,1>.

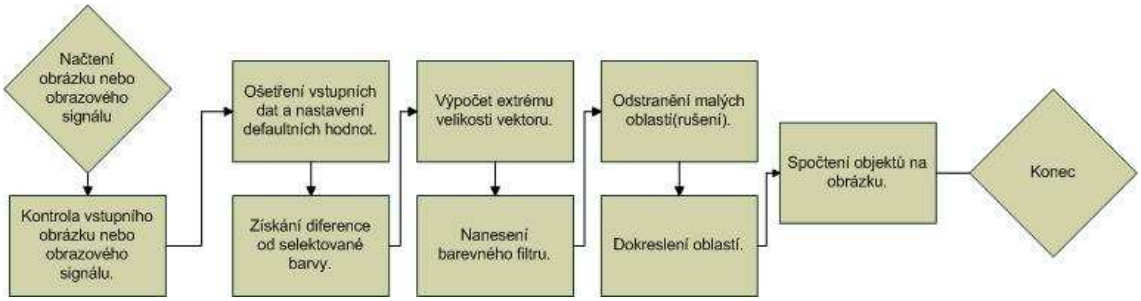
Následně je obrázek upraven pro sečtení jeho oblastí. Z matice diference pomocí prahu (na prazích závisí kvalita rozpoznání objektů) je vytvořeno logické pole a jsou odstraněny malé shluky černých pixelů (buněk s log. 1). Tímto způsobem je odstraněno rušení a šum. Dále je zjištěn počet pixelů největší a nejmenší oblasti a odstraní se příliš malé pixely (citlivost opět udává práh).

Nakonec se vyplní všechny uzavřené oblasti, aby se předešlo vícenásobnému započítání jednoho objektu.

Sečtení oblastí je provedeno pomocí dvou vnořených cyklů viz výstupní parametry (counter) níže.

Poté se prochází jednotlivé pixely, pokud se narazí na objekt (identifikovaný jako log. 1), inkrementuje se počítadlo a objekt se vyplní log. 1 pomocí záplavové výplně. Tímto se každý objekt započítá jen jednou. Pokud jsou stejně barevné objekty vedle sebe tak se započítají jako jeden. Tomu je zabráněno zvýšením barevného prahu Kvalitu rozpoznání by bylo možné nadále zvětšit filtrování barev místo prahu aplikovaného na rozdíl barevného vektoru a vektoru zvolené barvy, vyhodnocováním diferenciálů sousedních pixelů. Tím by byla zajištěna i jednoduchá detekce hran předmětů (detekce hran je používána u vektorové detekce geometrických prvků).

Přes svou jednoduchost tento program dokáže docela přesně spočítat počet objektů na obrázku. Po jistých úpravách by bylo možno jej využít i k realtimovému počítání např. vyrobených kusů lentilek na pásu za pomoci kamery.



Obr. 8. Blokové schéma detekce geometrických prvků pomocí barev.

V první řadě je z kamery načten obrazový signál. Následně je tento signál zkontrolován, neboť aplikace podporuje zpracování jen RGB signálů. Poté jsou ošetřeny vstupní data a to tak, že se nastaví na defaultní (počáteční) hodnoty. Tyto hodnoty lze samozřejmě nastavit i ručně v kódu. Dále je získána diference od vybrané barvy. Následuje výpočet extrému velikosti vektoru a je nanesen barevný filtr. Odstraní se rušení a dokreslí se oblasti, aby se sledované objekty započítaly pouze jednou. V posledním kroku jsou geometrické prvky v obrazovém signálu spočteny.

3.1.1 Implementace algoritmu pro detekování geometrických tvarů pomocí barev

Jako vstupní parametry aplikace pro detekci geometrických tvarů pomocí barev jsou zvoleny tyto vstupní parametry: ImData, colorVector, colorTreshold, regionTreshold.

ImData je parametr vstupních dat (obrazový signál).

```
imData = imread('C:\Users\Frey\Desktop\Bakalarskaprace\l1.jpg');
```

ColorVector je vektor definující hledanou barvu, pokud není zadána, pak se zobrazí dialog pro výběr barvy (pro přeskočení použijeme „ []“).

```
colorVector = [];
```

ColorTreshold je treshold barvy (barevná tolerance) není-li zadáno, použije se výchozí hodnota.

```
colorTreshold = 0.1;
```

RegionTreshold je treshold velikosti filtrovaných oblastí není-li zadáno, použije se výchozí hodnota.

```
regionTreshold = 0.3;
```

Jako výstupní parametry aplikace pro detekci geometrických tvarů pomocí barev jsou zvoleny tyto výstupní parametry: counter, diffMap, imMap.

Counter je proměnná do, které se ukládá počet spočtených objektů.

```
imData = ~imData(:,:);

imMap  = imData;

counter = 0;
```

Counter je použit v této dvojité smyčce, která slouží pro spočtení objektů v obraze.

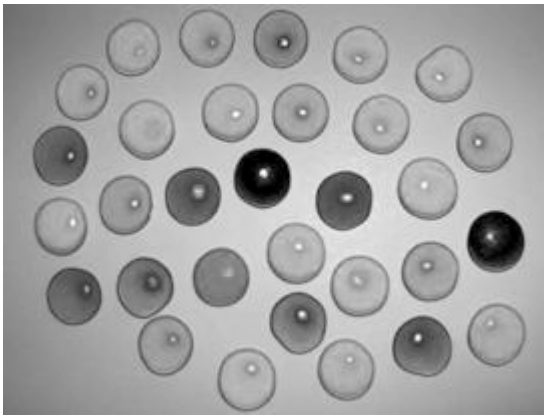
```
for cntH = 1:imgH

    for cntW = 1:imgW

        if imData(cntH,cntW) == 0

            counter = counter + 1;

            imData = imfill(imData, [cntH cntW], 8);

        end

    end

end
```

diffMap je to mapa (pole) s hodnotami 0 až 1definující barevnou shodu.

imMap je mapa (pole) zpracovaného obrázku.

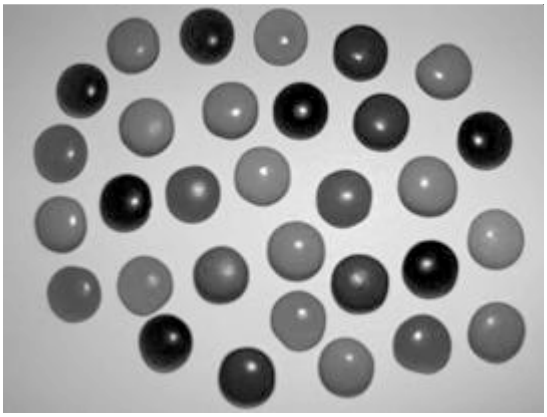
3.1.2 Výsledky implementace algoritmu pro detekování geometrických prvků pomocí barev

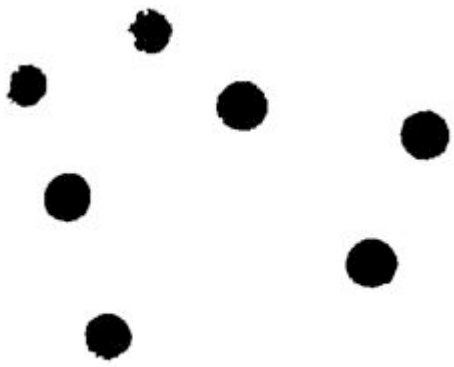
Obr. 9. Původní obraz.



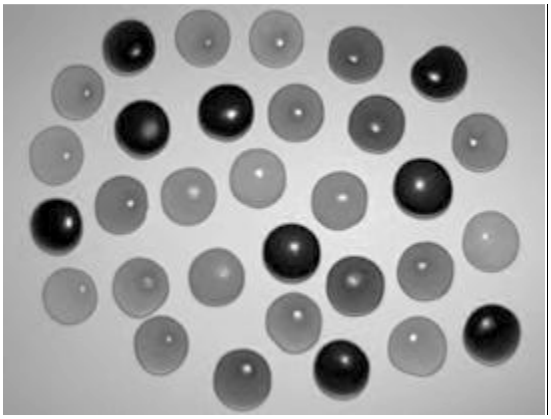
Obr. 10. Binarizovaný obraz podle modré barvy.

Obr. 11. Detekované modré kruhové objekty.

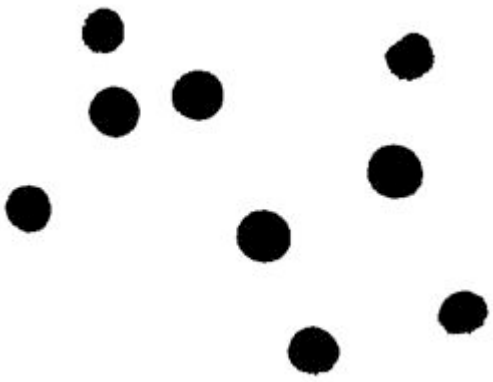




Obr. 12. Binarizovaný obraz podle červené barvy. Obr. 13 Detekované červené kruhové obj.



Obr. 14. Binarizovaný obraz podle žluté barvy.



Obr. 15. Detekované žluté kruhové objekty.

3.2 Vektorová detekce geometrických prvků

Vektorové detekci předchází celá řada metod. Obraz je nejprve filtrován pomocí Gaussovkého filtru. Následně je provedena binarizace obrazu, která obraz převede na černobílý, odstraní dodatečné šumy a obraz zaostří. Poté jsou prvky v obraze převedeny na šířku jednoho pixelu. Toto zajišťuje metoda skeletonizace. Dále jsou odděleny jednotlivé znaky v obrazu – je provedena segmentace. Jednotlivé znaky jsou převedeny na vertexy a ty jsou pospojovány přímkami. Výsledek vektorizace a výsledky jednotlivých dílčích metod jsou zobrazeny v jednotlivých oknech aplikace.

Nejprve jsou uvedeny metody, které předcházejí samotné detekci geometrických prvků z obrazového signálu.

3.2.1 Rozbor filtrace pomocí Gaussovké funkce

Gaussovská filtrace je výsledkem rozmazání obrazu pomocí Gaussovké funkce. Ta to metoda je velice rozšířená v technice zpracování obrazu. Gaussovká filtrace snižuje obrazový šum, ale také detaily obrazu. Vizuální efekt tohoto rozostření se podobá prohlížení obrázku přes průsvitné plátno.

Matematické použití Gaussovké filtrace na obraz je stejné jako konvoluce obrazu s Gaussovkou funkcí.

Gaussovký filtr je typ filtru, který používá Gaussovu funkci pro výpočet transformace. Tato transformace je použita na každý pixel filtrovaného obrazu. Rovnice Gaussovi funkce v jednom rozměru je:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \quad (2)$$

Ve dvou rozměrech to je:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (3)$$

Kde x je vzdálenost od počátku na vodorovné ose, y je vzdálenost od počátku na vertikální ose a σ je směrodatná odchylka Gaussového rozložení (čím větší je, tím více je obraz rozmazán).

Při aplikaci ve dvou rozměrech tento vzorec vytváří povrch, jehož obrysy jsou soustředné kružnice s Gauusovskou distribucí od středu. Hodnoty z této distribuce se používají k vytvoření konvoluční matice, která se aplikuje na původní obraz. Nová hodnota pixelu je nastavena na vážený průměr okolí toho pixelu. Původní hodnota pixelu obdrží největší váhu (má největší Gaussovkou hodnotu) a sousední pixely dostávají menší váhu, jak se jejich vzdálenost k původnímu pixelu zvětšuje. To má za následek rozmazání, které chrání hranice a hrany lépe než ostatní rovnoměrně rozmazávající filtry.

V zásadě bude Gaussova funkce v každém bodě nenulová, což znamená, že celý obraz bude muset být zahrnutý do výpočtu pro každý pixel.

V praxi při výpočtu diskrétní aproximace Gaussovou funkcí se pixely ve vzdálenosti více než 3 σ jsou dostatečně malé, aby se daly považovat za nuly. To znamená, že příspěvek pixelů mimo rozsah může být ignorován. Tak že program pro zpracování obrazu potřebuje vypočítat pouze matici o rozměrech $\lceil 6\sigma \rceil x \lceil 6\sigma \rceil$ (kde $\lceil\ \rceil$ je hranice funkce) aby byl výsledek dostatečně podobný tomu získanému celou Gaussovou distribucí.

Kromě toho, že je kruhově symetrické, Gaussovkou filtraci lze aplikovat na dvourozměrný obraz jako dva nezávislé jednorozměrné výpočty proto se mu také říká, že je lineárně oddělitelné.

Efektu dvourozměrné matice lze dosáhnout použitím řady jednorozměrných Gaussových matic v horizontálním směru a pak opakovat postup ve vertikálním směru. [9]

3.2.2 Implementace algoritmu pro Gaussovkou filtraci

V první řadě je vytvořeno jádro pro Gaussovkou filtraci, která obsahuje dva parametry intenzitu a vzdálenost. Intenzitou σ je určena míra rozmazání obrazu. Tento parametr lze v aplikaci libovolně měnit. Vzdálenost je vzdálenost od počátku na vodorovné ose a od počátku na vertikální ose viz vzorec výše.

```
public static double[] JadroRozmazani(double intenzita, int vzdalenost)
        {
            double[] jadro = {1};
            double normalizaceJadra = 0;
```

V této smyčce je nově vytvořené jádro naplněno Gaussovkou funkcí. Jde prakticky o přepis vzorce Gaussovské funkce do jazyka C#. Po naplnění jádra Gaussovkou funkcí je jádro normalizováno.

```
for (int i = 0; i < vzdalenost * 2 + 1; i++)
            {
jadro[i] = 1 / (Math.Sqrt(2 * Math.PI) * intenzita) * Math.Exp(-(i - vzdalenost)
* (i - vzdalenost) / (2 * intenzita * intenzita));
                normalizaceJadra += jadro[i];
            }
```

V následující části je opět vytvořeno jádro. Nicméně toto jádro už vytvořeno pro samotné Gaussovké rozmazání a má přednastavenou intenzitu rozmazání σ = 3.

```
public static double[] JadroRozmazani(double intenzita)
        {
return JadroRozmazani(intenzita, (int)Math.Ceiling(intenzita * 3));
        }
```

V dalším kroce se provede Gaussovské rozmazání s přednastavenou intenzitou.

```
public static bool Rozmazani(ref Bitmap obrazek, double intenzita)
        {
return Rozmazani(ref obrazek, JadroRozmazani(intenzita));
        }
```

Pokud není obrázek ve formátu 32b s barvami Alpha + R + G + B musí se převést. Převod se provede následujícím způsobem. Vytvoří se obrázek s určitou šířkou a výškou, který už je v požadovaném formátu. A původní obrázek se na něj převede.

```
if (obrazek.PixelFormat != PixelFormat.Format32bppArgb)
            {
Bitmap     prevodObrazek     =     new     Bitmap(obrazek.Width,     obrazek.Height,
PixelFormat.Format32bppArgb);
Graphics g = Graphics.FromImage(prevodObrazek);
g.DrawImage(obrazek, new Point(0, 0));
g.Dispose();
obrazek = prevodObrazek;
            }
```

Dále se získají data s převedeného obrázku (výška, šířka, formát).
Vytvořené jádro je nejprve aplikováno ve vertikálním směru. A to tímto způsobem – původní obrázek => následuje původní obrázek => a pomocný obrázek.

```
for (int radek = 0; radek < dataPomocnehoObrazku.Height; radek++)
                {
                    for    (int    sloupecPomocny    =    0;    sloupecPomocny    <
dataPomocnehoObrazku.Width; sloupecPomocny++)
                    {
```

Následně se projde všemi prvky jádra. Tyto prvky se aplikují na korespondující pixely obrázku a přičtou se k pomocnému obrázku. Sčítání je provedeno po barevnách prvcích, první se přičte modrá

barva pak zelená barva dále červená a nakonec alpha kanál. Alpha kanál nese informaci o průhlednosti.

```
for (int jadroIndex = 0; jadroIndex < jadro.GetLength(0); jadroIndex++)
                        {
                                                                int       sloupec       =
sloupecPomocny + jadroIndex - jadro.GetLength(0) / 2;
if (sloupec < 0 || sloupec >= dataObrazku.Width)
                                    continue;
 byte* dataPixelu = (byte*)dataObrazku.Scan0 + radek * dataObrazku.Stride +
sloupec * 4;
dataPixeluPomocny[0] += (byte)(dataPixelu[0] * jadro[jadroIndex]); //B
dataPixeluPomocny[1] += (byte)(dataPixelu[1] * jadro[jadroIndex]); //G
dataPixeluPomocny[2] += (byte)(dataPixelu[2] * jadro[jadroIndex]); //R
dataPixeluPomocny[3] += (byte)(dataPixelu[3] * jadro[jadroIndex]); //A
                        }
```

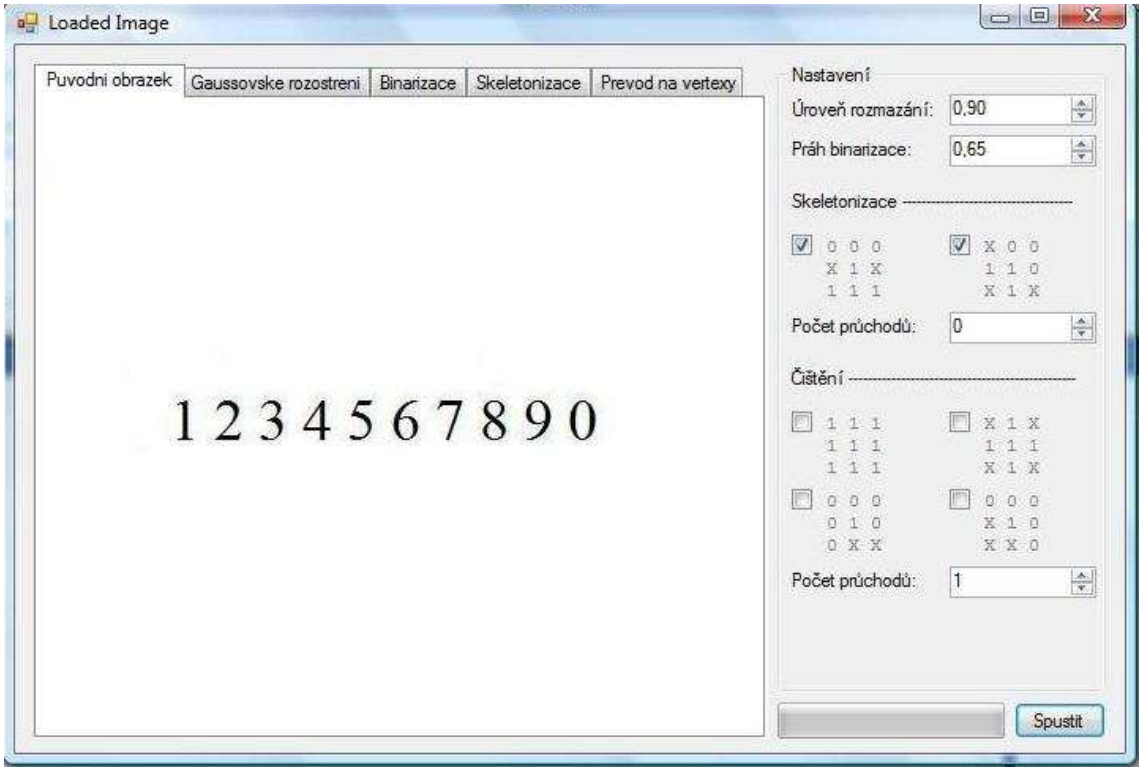
V dalším kroce je jádro aplikováno v horizontálním směru podle Gaussovké funkce. A to tímto způsobem – pomocný obrázek => horizontální rozmazání => výsledný obrázek.

```
for (int radek = 0; radek < dataObrazku.Height; radek++)
                {
for (int sloupec = 0; sloupec < dataObrazku.Width; sloupec++)
                    {
```

A pak se algoritmus opakuje, projde se prvky jádra. Prvky se aplikují na korespondující pixely pomocného obrázku a jsou k němu přičteny.
Gaussovké filtrace se provádí podle metody, která je uvedená výše. Je použito dvou jednorozměrných Gaussovských matic, které jsou aplikovány v horizontálním a vertikálním směru.

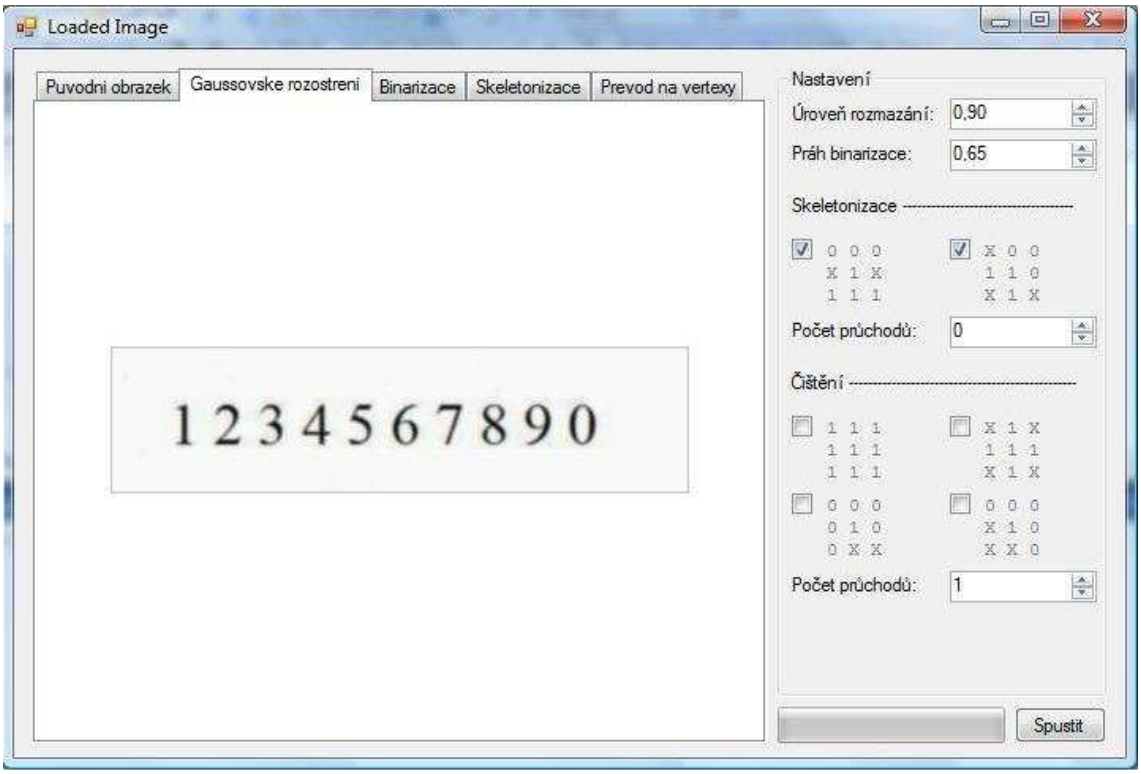
3.2.3 Výsledky implementace algoritmu pro Gaussovskou filtraci obrazu

Na Obr. 16. Je zobrazen původní obraz.



Obr. 16. Původní obraz.

Následně je obraz rozmazán pomocí metody Gaussovského filtrování s přednastavenou směrodatnou odchylkou $\sigma = 0{,}90$. Směrodatná odchylka určuje úroveň rozmazání.



Obr. 17. Gaussovky filtrovaný obraz.

3.3.1 Rozbor binarizace obrazu

Pro potlačení šumu se často zařazuje dodatečná fáze binarizace, kdy všechny pixely obrazu rozdělíme do dvou skupin -- pixely objektů (označené např. černě a s hodnotou 1) a pixely pozadí (bílé s hodnotou 0). Pak je možné celou mapu představující obraz komprimovat a vždy 8 pixelů uložit do 1 bytu. [10]

Binarizace změní obraz, který je v šedé škále nebo v barvě na černobílý. Pomocí parametrů nastavení hloubky a detailů lze odstranit z obrazu některé nečistoty.

3.3.2 Implementace algoritmu pro binarizaci obrazu

V první řadě je obraz převeden na černobílý (2 barvy) podle prahu jasu (0-1):

```
public static void Prah(ref Bitmap obrazek, double prah)
```

Pokud není obraz ve formátu 32b s barvami Alpha + R + G + B musí se převést. Převod se provede následujícím způsobem. Vytvoří se obrázek s určitou šířkou a výškou, který už je v požadovaném formátu. A původní obraz se na něj převede. Tato část je také využita v algoritmu pro Gaussovké rozmazání.

```
if (obrazek.PixelFormat != PixelFormat.Format32bppArgb)
            {
Bitmap    pomocnyObrazek    =    new    Bitmap(obrazek.Width,    obrazek.Height,
PixelFormat.Format32bppArgb);
Graphics g = Graphics.FromImage(pomocnyObrazek);
g.DrawImage(obrazek, new Point(0, 0));
g.Dispose();
obrazek = pomocnyObrazek;
            }
```

Dále se získají data obrazu (šířku, výšku a jeho formát).

```
BitmapData  dataObrazku  =  obrazek.LockBits(new  Rectangle(0,  0,  obrazek.Width,
obrazek.Height), ImageLockMode.ReadWrite, PixelFormat.Format32bppArgb);
```

Následující smyčky slouží k průchodu obrazem. Poté se získá ukazatel na pixel, se kterým se dále pracuje.

```
for (int i = 0; i < dataObrazku.Height; i++)
                {
                    for (int j = 0; j < dataObrazku.Width; j++)
                    {
byte* dataPixelu = (byte*)dataObrazku.Scan0 + i * dataObrazku.Stride + j * 4;
```

V další části se získá jas pixelu což je v podstatě průměr z {R; G; B}.

```
double    jasPixelu    =    ((double)dataPixelu[0]    +    (double)dataPixelu[1]
+(double)dataPixelu[2]) / (3.0 * 256);
```

Následuje podmínka, která zajišťuje následující. Pokud je jas menší než práh pak se pixel nastaví na černý.
Podmínka obsahuje dodatek, který vyjadřuje druhou možnost. Pokud není jas menší, než práh pak se pixel nastaví na bílý.

```
if (jasPixelu < prah)
                        {
                            dataPixelu[0] = 0;
                            dataPixelu[1] = 0;
                            dataPixelu[2] = 0;
                        }
 else
                        {
                            dataPixelu[0] = 255;
                            dataPixelu[1] = 255;
                            dataPixelu[2] = 255;
                        }
```

Bitové pole se převede na černobílý obraz a vytvoří se nový obrázek o velikosti toho bitového pole, který je ve formátu s barvami RGB aplha.

```
public static Bitmap bity2obrazek(ref bool[][] bityObrazku)
Bitmap obrazek = new Bitmap(bityObrazku[0].Length, bityObrazku.GetLength(0),
PixelFormat.Format32bppArgb);
```

Opět se získají data obrazu (šířka, výška a formát). Pomocí dvou smyček for se projde obrazem a získá se ukazatel na pixel, se kterým budeme pracovat.

```
BitmapData dataObrazku = obrazek.LockBits(new Rectangle(0, 0, obrazek.Width,
obrazek.Height), ImageLockMode.WriteOnly, PixelFormat.Format32bppArgb);
for (int i = 0; i < dataObrazku.Height; i++)
for (int j = 0; j < dataObrazku.Width; j++)
byte* dataPixelu = (byte*)dataObrazku.Scan0 + i * dataObrazku.Stride + j * 4;
```

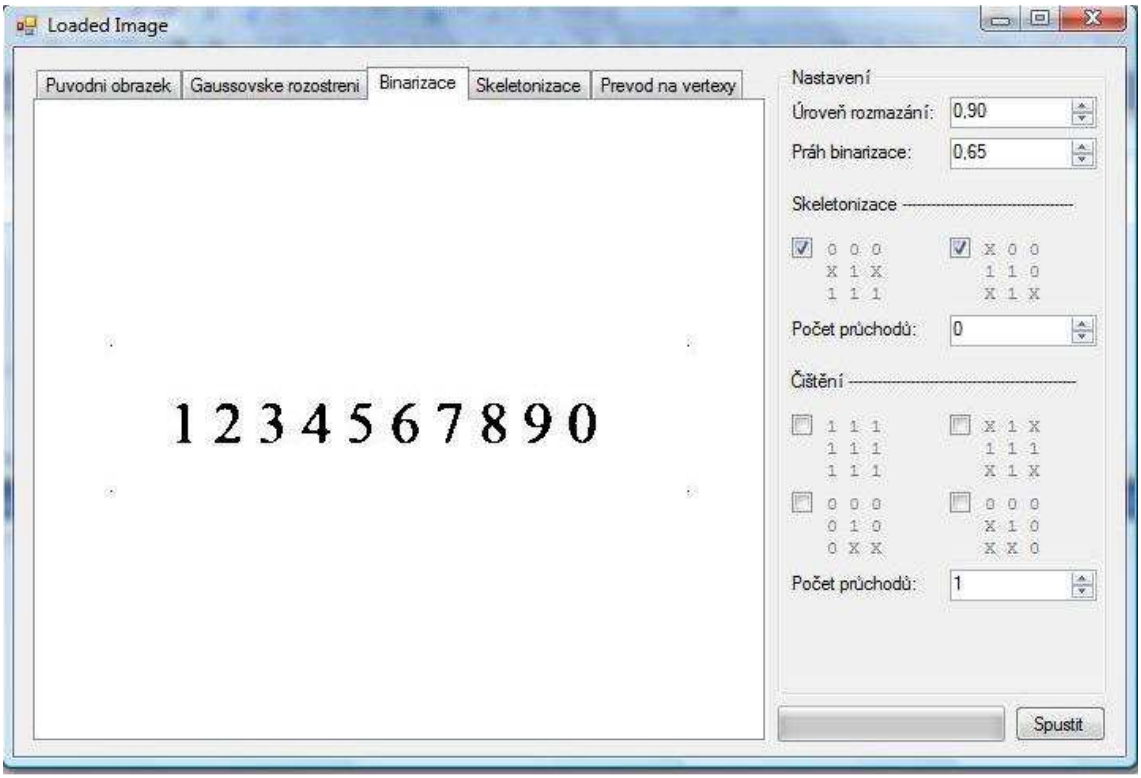
Další podmínka zaručuje, že pokud je bit v bitovém poli TRUE (pravdivý tzn. 1) nastaví se aktuální pixel obrazu do černé barvy.
Podmínka obsahuje dodatek, který vyjadřuje opačnou situaci. Takže pokud je bit v bitovém poli FALSE (nepravdivý tzn. 0) aktuální pixel se nastaví do bíle barvy.

```
if (bityObrazku[i][j])
                        {
                            dataPixelu[0] = 0;
                            dataPixelu[1] = 0;
                            dataPixelu[2] = 0;
                            dataPixelu[3] = 0xff;
                        }
else
                        {
                            dataPixelu[0] = 0xff;
                            dataPixelu[1] = 0xff;
                            dataPixelu[2] = 0xff;
                            dataPixelu[3] = 0xff;
                        }
```

Po binarizaci je obraz vyčištěn, zaostřen a převeden na černobílý. Je připraven pro další zpracování.

3.3.3 Výsledky implementace algoritmu pro binarizaci obrazu

Binarizovaný obraz s přednastavenou hodnotu práhu binarizace 0,65.



Obr. 18. Binarizovaný obraz.

3.4.1 Rozbor skeletonizace obrazu

Ztenčení je morfologická operace, která se používá k odstranění vybraných pixelů s popředí binárních obrazů. Může být použito pro celou řadu aplikací, ale předně se používá pro metodu skeletonizace. V této formě je ztenčení běžně používáno k vyčistění výstupu s detekce hran snížením všech čar na jednopixelovou šířku. Ztenčení se zpravidla vztahuje pouze na binární obrazy a vytváří další binární obraz jako výstup.

Binární strukturální prvky používané k ztenčování jsou rozšířený typ hit-and-miss transformace (tj. mohou obsahovat jak jedničky, tak nuly).

Operace ztenčení souvisí s hit-a-miss transformací a může být vyjádřena docela jednoduše. Ztenčení obrazu I pomocí strukturovaného prvku J je:

$$thin(I,J) = I - hit - and - miss(I,J) \quad (4)$$

Kde odčítání je logické odčítání definované:

$$X - Y = X \cap NOT\ Y \quad (5)$$

Operace ztenčení je vypočtena překladem počátku strukturovacího prvku do každé možné polohy pixelu v obraze. A v každé takové pozici se porovnává se základními obrazovými pixely.

Pokud pixely v popředí a v pozadí obrazu ve strukturovaném prvku přesně odpovídají pixelům v popředí a v pozadí původního obrazu. Pak pixel, který je, pod zdrojem strukturovaného prvku je zasazen do pozadí (nula). Jinak je obraz nemění. Všimněme si, že strukturovaný prvek musí mít ve svém počátku vždy hodnotu jedna nebo nic (x), aby tato metoda fungovala. Výběr strukturovaného

prvku určuje, za jakých okolností bude pixel v popředí zasazen do pozadí, a tudíž určuje aplikaci pro operaci ztenčení.

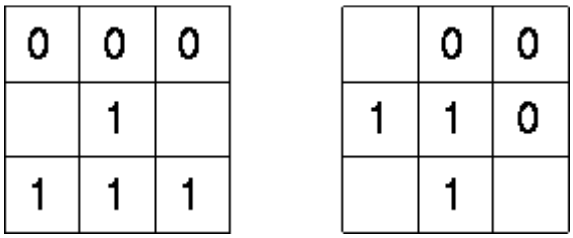
Ztenčení se nejčastěji používá na úpravu signálu vystupujícího s detektoru hran. Obraz je ztenčen na tloušťku jednoho pixelu při zachování původní délky čar, ze kterých se obraz skládá.

Na to je použit tento jednoduchý algoritmus:

Uvažovány jsou všechny pixely na hranicích v oblasti popředí (tj. body v popředí, které mají alespoň jeden sousední bod v pozadí). Následně je smazán každý bod, který má víc než jeden sousední bod v popředí. Toto lze provést jen tehdy, když se smazáním nepřeruší (tj. rozdělí se na dvě části) oblast obsahující pixel. Toto se opakuje až do konvergence. Tato procedura narušuje hranice objektů v popředí jak jen to je možné, ale nemá žádný vliv na pixely na koncích čar.

Tohoto efektu lze dosáhnout morfologických ztenčováním pomocí iterace až do konvergence se strukturálními prvky podle obrázku 13 a všech jejich $90°$ rotací (celkem $4x2 = 8$ strukturálních prvků).

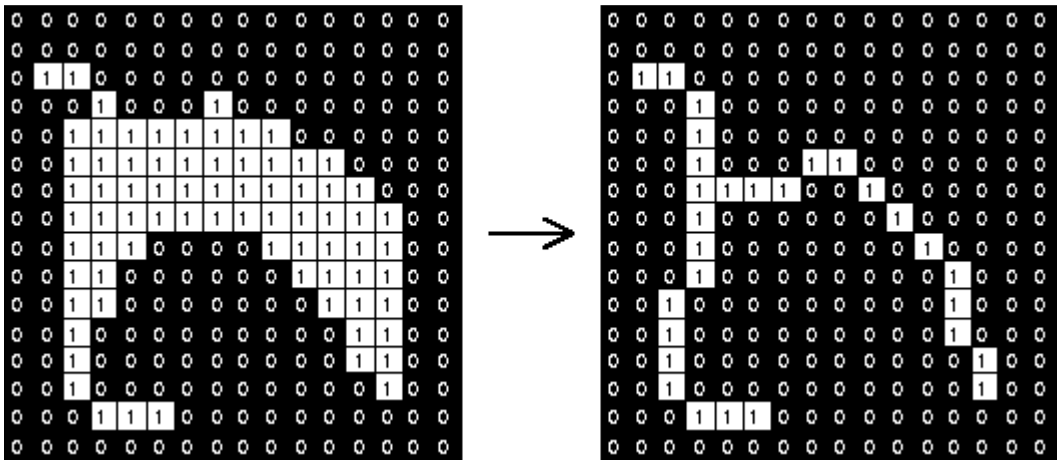
Ve skutečnosti to co je zde stanoveno je octagonální (osmiboká) kostra binárního tvaru – soubor bodů, které leží ve středech octangonů. Všechny tyto body se vlezou do octagonálního tvaru a dotýkají se hran nejméně ve dvou bodech. Tato metoda garantuje vyprodukování nepřerušené kostry. [12] [11]



Obr. 19. Strukturované prvky [11]

Obrázek 19. Strukturované prvky pro skeletonizaci pomocí morfologického ztenčování. V každé iteraci je obraz napřed ztenčen pomocí levého strukturovacího prvku a následně pomocí pravého a pak šesti zbývajícími $90°$ rotacemi dvou prvků. Tento proces je cyklicky opakován do té doby kdy už ztenčování nemá žádný efekt. Jako vždy je zdroj strukturovaných prvků ve středu obrazu. [12] [11]

Výsledek ztenčovaní na jednoduchém binárním obrázku.



Obr. 20. Obrázek morfologického ztenčování za použití strukturovacích elementů uvedených na obrázku 19. [11]

3.4.2 Implementace algoritmu pro skeletonizaci obrazu

Výčet možných strukturovaných prvků masky jádra, které jsou používané pro skeletonizaci. Tyto prvky jsou jedna, nula a nic (x).

```
public enum maska
    {
        _0,
        _1,
        _X
    }
```

Dále jsou vyčteny všechny směry otočení používaných jader. Jádra se mohou otočit o $0°$, $90°$, $180°$ a $270°$.

```
public enum otoceni
    {
        _0deg,
        _90deg,
        _180deg,
        _270deg
    }
```

Následně je vytvořena třída skeletonizace, která obsahuje seznam jader. Konstruktor pro vytvoření jádra a samotné přidání jádra. Třída také obsahuje metodu pro otočení jádra.

```
public class Skeletonizace
    {
private List<maska[][]> jadra;
public Skeletonizace()
        {
            jadra = new List<maska[][]>();
        }
public void Pridat(maska[][] jadro)
        {
            jadra.Add(jadro);
private maska[][] OtocitJadro(maska[][] jadro, otoceni otoceni)
```

Následuje podmínka, která zajišťuje, že pokud se má jádro otočit o $0°$ tak se vrátí nezměněné.

```
if (otoceni == otoceni._0deg)
                return jadro;
```

Vytvoření nového pole pro otočené jádro.

```
maska[][] otoceneJadro = new maska[3][];
for (int i = 0; i < otoceneJadro.GetLength(0); i++)
otoceneJadro[i] = new maska[3];
```

Původní jádro strukturovaných prvků se otočí o $90°$.

```
case otoceni._90deg:
                    otoceneJadro[0][0] = jadro[2][0];
                    otoceneJadro[0][1] = jadro[1][0];
                    otoceneJadro[0][2] = jadro[0][0];
                    otoceneJadro[1][0] = jadro[2][1];
                    otoceneJadro[1][1] = jadro[1][1];
                    otoceneJadro[1][2] = jadro[0][1];
                    otoceneJadro[2][0] = jadro[2][2];
                    otoceneJadro[2][1] = jadro[1][2];
                    otoceneJadro[2][2] = jadro[0][2];
                    break;
```

Následně se jádro otočené o $90°$ otočí o dalších $90°$ tzn., že se jádro v konečném výsledku otočí o $180°$.

```
case otoceni._180deg:
                    otoceneJadro[0][0] = jadro[2][2];
                    otoceneJadro[0][1] = jadro[2][1];
                    otoceneJadro[0][2] = jadro[2][0];
                    otoceneJadro[1][0] = jadro[1][2];
                    otoceneJadro[1][1] = jadro[1][1];
                    otoceneJadro[1][2] = jadro[1][0];
                    otoceneJadro[2][0] = jadro[0][2];
                    otoceneJadro[2][1] = jadro[0][1];
                    otoceneJadro[2][2] = jadro[0][0];
                    break;
```

Jádro otočené o $180°$ se otočí o dalších $90°$ tzn., že se jádro v konečném výsledku otočí o $270°$.

```
case otoceni._270deg:
                    otoceneJadro[0][0] = jadro[0][2];
                    otoceneJadro[0][1] = jadro[1][2];
                    otoceneJadro[0][2] = jadro[2][2];
                    otoceneJadro[1][0] = jadro[0][1];
                    otoceneJadro[1][1] = jadro[1][1];
                    otoceneJadro[1][2] = jadro[2][1];
                    otoceneJadro[2][0] = jadro[0][0];
                    otoceneJadro[2][1] = jadro[1][0];
                    otoceneJadro[2][2] = jadro[2][0];
                    break;
```

Poté se testuje, jestli se maska (bod jádra) shoduje s bodem obrázku. Pokud je maska 0 bod i je také 0. Pokud je maska 1 bod i je také 1. Všechny ostatní možnosti platí.

```
private bool TestMaskyNaBod(maska maska, bool bod)
        {
switch (maska)
            {
                case maska._0:
                    return !bod;
                case maska._1:
                    return bod;
            }
            return true;
            }
```

Test jestli se jádro shoduje s bodem obrazu.

```
private bool AplikovatJadroNaBod(maska[][] jadro, ref bool[][] bityObrazku, int
X, int Y)
        {
```

Dále se projde přes řádky obrazu. A testuje se podmínka, pokud je bod mimo obrázek pak je vždy platný. Projde se přes sloupce obrazu. A opět se testuje podmínka, pokud je bod mimo obrázek je pak vždy platný. Následuje podmínka, která testuje, jestli se maska (bod jádra) shoduje s bodem obrázku.

Následně se aplikují všechna jádra ze seznamu na obraz. Opět se projde obrazem a jsou aplikována další jádra. Pokud se jádro shoduje s bodem obrázku tak se bod vynuluje.

Aplikace skeletonizace s jádry v seznamu (provádí se, dokud se obrázek mění).

```
public void AplikovatSkeletonizaci(ref bool[][] bityObrazku)
        {
            this.AplikovatSkeletonizaci(ref bityObrazku, 0);
```

Aplikace skeletonizace dokud se obrázek mění, nebo už byl dosažen maximální počet aplikací:

```
for (int i = 0; zmena && (maxPocetAplikaci <= 0 || i < maxPocetAplikaci); i++)
            {
```

Následně se aplikuje otočení všech jader a to v tomto pořadí. První se aplikuje otočení o $0°$. Poté se aplikuje otočení o $90°$. Dále se aplikuje otočení o $180°$. V poslední řadě se aplikuje otočení o $270°$.

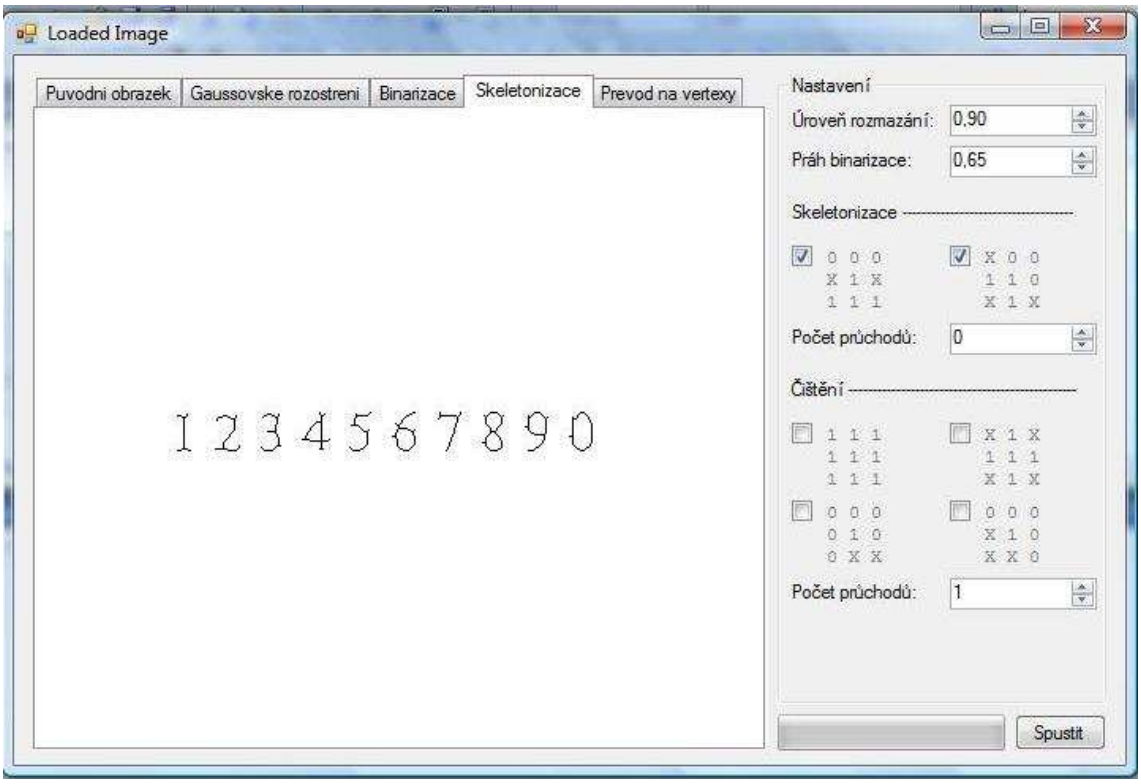
```
                zmena = false;
                zmena |= this.AplikovatJadra(ref bityObrazku, otoceni._0deg);
                zmena |= this.AplikovatJadra(ref bityObrazku, otoceni._90deg);
                zmena |= this.AplikovatJadra(ref bityObrazku, otoceni._180deg);
                zmena |= this.AplikovatJadra(ref bityObrazku, otoceni._270deg);
```

Obraz je ztenčen na velikost jednoho pixelu a skeletizován (obraz je převeden na spojitou kostru).

3.4.3 Výsledky implementace algoritmu pro skeletonizace obrazu

Skeletonizovaný obraz pomocí strukturovaných prvků viz Obr. 16. V uživatelském rozhraní je nastaven počet průchodů na 0, nicméně implicitně se projde obrazem jednou. V položce čistění lze

použít další čtyři jádra strukturovaných prvků. Ale tyto jádra nejsou vhodné ke skeletonizaci obrazu s čísly.



Obr. 21. Skeletonizovaný obraz.

3.5.1 Rozbor převodu obrazu na vertexy a jeho následná segmentace

Vertex je v počítačové reprezentaci grafická struktura, která popisuje bod ve 2D nebo ve 3D prostoru. Většinu vlastností vertexu reprezentují vektory v prostoru, kde mají být vykresleny. Obraz je nejprve převeden na vertexy a poté je segmentován to znamená, že jsou odděleny jednotlivé znaky. [14]

V oblasti digitálního zpracování obrazu, se segmentace odkazuje na proces dělení digitálního obrazu do více segmentů. Cílem segmentace je zjednodušit nebo změnit reprezentaci obrazu na něco co je smysluplnější a snadněji analyzovatelné než je původní obraz. Segmentace obrazu se obvykle používá k nalezení objektů a jejich hranic v obraze. Přesněji řečeno segmentace obrazu je proces přiřazování označení každému pixelu v obraze, tak že pixely ze stejnou značkou sdílí některé obrazové charakteristiky.

Výsledek segmentace obrazu je množina segmentů, které společně pokrývají celý obraz. Každý z pixelů v oblasti je podobný s ohledem na některé charakteristické znaky, jako jsou: barva, intenzita, textura. Přilehlé oblasti se výrazně liší, pokud jde o stejné charakteristiky.

Typickým cílem segmentace obrazu je identifikace popředí a určení oblastí v obraze odpovídajícím významnému prvku zachycené scény. [13]

3.5.2 Implementace algoritmu pro převod obrazu na vertexy

V první řadě je vytvořen seznam vertexů.

```
        private List<Point> vertexy = new List<Point>();
```

Následně je bitové pole obrazu převedeno na vertexy.

```
public void bityObrazku2vertexy(ref bool[][] bityObrazku)
        {
```

Následují dvě smyčky, které zajišťují průchod přes všechny bity v obraze.

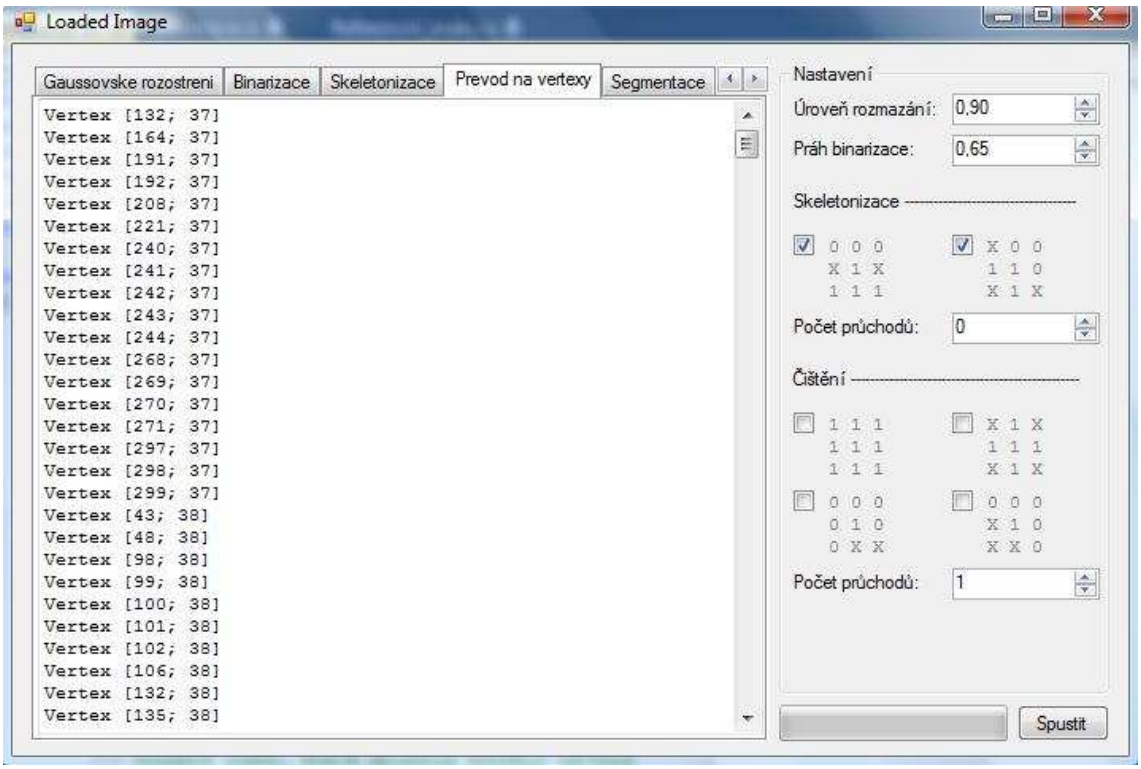
```
for (int i = 0; i != bityObrazku.GetLength(0); i++)
            {
for (int j = 0; j != bityObrazku[0].Length; j++)
                {
```

Pokud je bit v jedničce, uloží se jako vertex.

```
if (bityObrazku[i][j])
this.vertexy.Add(new Point(j, i));
```

3.5.3 Výsledky implementace algoritmu pro převod obrazu na vertexy

Obraz převedený na souřadnice jednotlivých bodů, ze kterých se prvky v obrazu skládají.



Obr. 22. Obraz převedený na vertexy.

3.5.4 Implementace algoritmu pro segmentaci obrazu

V první řadě je určena maximální vzdálenost mezi vertexy. Což je prakticky jen nastavená konstanta.

```
private const float maximalniVzdalenost = 2.0f;
```

Ze seznamů vertexů se vytvoří seznam symbolů a vertexy jsou překopírovány do dočasného seznamu.

```
List<Point> vertexes = new List<Point>(_vertexes);
```

Vertexy jsou seřazeny podle osy X. Jako parametr metody sort je vložena pomocí příkazu lamda anonymní metoda, která porovnává Xové souřadnice bodů a vrací informaci o tom, jestli je bod menší větší nebo stejný.

```
vertexes.Sort(((Point p1, Point p2) =>
            {
                if (p1.X < p2.X) return -1;
                else if (p2.X < p1.X) return 1;
                else return 0;
            }));
```

Následně se budou přidávat symboly, dokud nezbude nějaký bod. Poté je zavolána funkce PridatSymbol, která se postará o přidání symbolu a načtení vyhovujících vertexů.

```
PridatSymbol(vertexes);
```

Dále se vykreslí jednotlivé symboly složené z bodů, které jsou barevně odděleny. Barvy pro jednotlivé znaky jsou voleny náhodně. K tomuto vykreslení slouží tato metoda.

```
public void VykreslitSymboly(Image img)
        {
            Graphics g = Graphics.FromImage(img);
            g.Clear(Color.White);
            Random rnd = new Random();
```

Následující smyčkou se projde všemi symboly.

```
foreach (List<Point> s in _symboly)
```

Po průchodu všemi symboly se nastaví náhodná barva pro jejich vykreslení.

```
Color drawCol = Color.FromArgb(rnd.Next() % 256, rnd.Next() % 256, rnd.Next() %
256);
```

Opět se projde všemi symboly a ty se vykreslí.

Poté jsou jednotlivé vertexy skládány rovnou do přímek podle nejbližšího bodu. Dále je vytvořen seznam získaných vertexů.

```
private void PridatSymbol(List<Point> vertexy)
        {
            List<Point> symbol = new List<Point>();
```

K seznamu vertexů je přidán další vertex podle, kterého se budou porovnávat ostatní.

```
symbol.Add(vertexy[0]);
```

Následující smyčka přidává vertexy do symbolu, dokud nezbude žádný bod. Tato smyčka je doplněna o čítač přidaných vertexů. Tento čítač slouží ke zrušení smyčky a tu zruší, pokud není nalezen žádný vyhovující vertex.

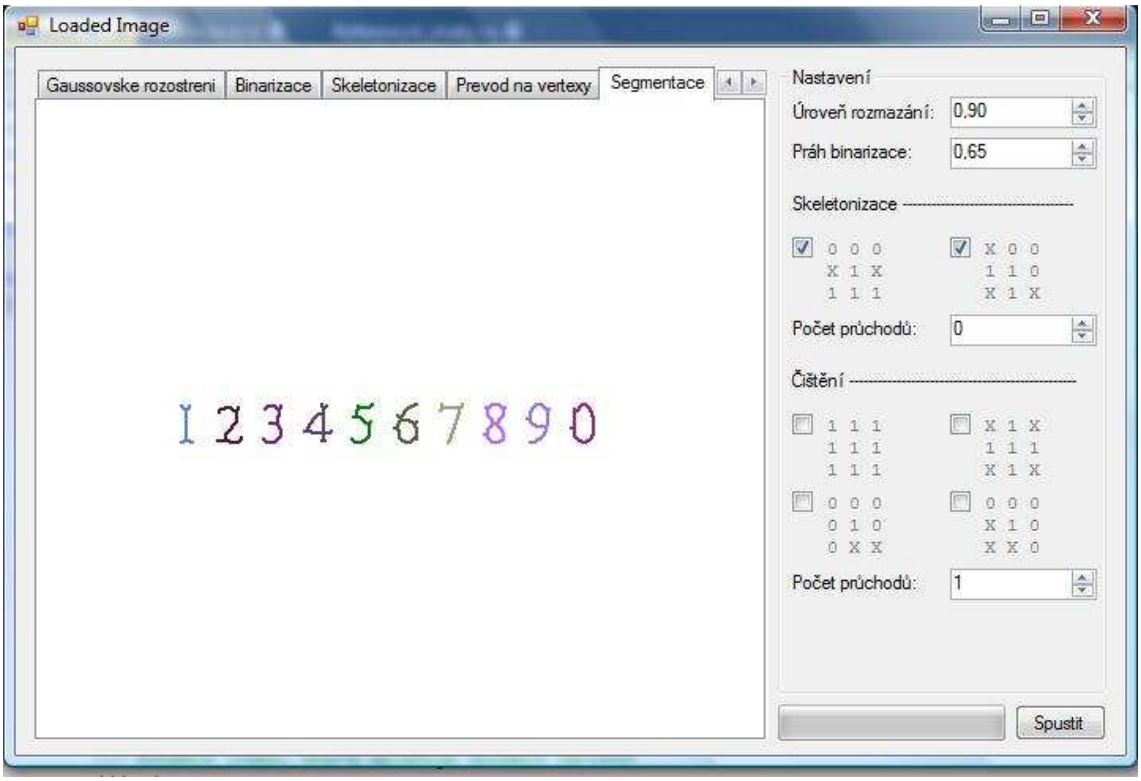
```
while (vertexy.Count != 0)
  {
                int added = 0;
                for (int y = 0; y < symbol.Count; y++)
                {
                    for (int i = 0; i < vertexy.Count; i++)
                    {
```

Jsou zjištěny všechny vertexy, které vyhovují a které jsme ještě nepřidali. Následující smyčka zjišťuje vzdálenost mezi už přidaným vertexem a dalším vertexem ze seznamu. Pokud je vzdálenost menší než je povolená vzdálenost tak přidáme vertex. A poté je inkrementován čítač přidaných vertexů. Tuto smyčku ošetřuje podmínka, pokud nebyl nalezen žádný nový vertex, který splňuje vzdálenost, tak smyčka se zruší.

Obraz je segmentován a jednotlivé znaky jsou barevně odděleny.

3.5.5 Výsledky implementace algoritmu pro segmentaci obrazu

Obraz je segmentován – jednotlivé prvky jsou barevně odděleny a následně převedeny úsečkami zpět z vertexů na symboly. Na to slouží následující metoda.



Obr. 23. Segmentovaný obraz.

3.6.1 Rozbor vektorizace obrazu

Vektorová grafika je jeden ze způsobů reprezentace obrazových informací. Ve vektorové grafice je obraz reprezentován použitím geometrických prvků, jako jsou: body, přímky, křivky, tvary nebo polygony, které jsou všechny založeny na matematických rovnicích. Dalším způsobem reprezentace obrazových informací je bitmapová grafika. V bitmapové grafice je celý obraz popsán pomocí jednotlivých barevných bodů (pixelů).

Jelikož je vektorový obraz vyjádřen matematicky lze s ním lépe manipulovat než s obrazem v bitmapové grafice. Například zvětšení, zmenšení, zakřivení a přeložení prvků v obrazu lze dosáhnout bez zkreslení či ztráty kvality. Také to znamená, že monitory s vyšším rozlišením zobrazí vektorizovaný obraz také s vyšším rozlišením. Kdežto obraz reprezentovaný bitmapovou grafikou má přednastavené rozlišení, které nelze překročit bez ztráty kvality obrazu. [15]

Popis algoritmu

Nejprve jsou založeny referenční seznamy s prvky, které jsou detekovány. Tyto prvky nejsou převedeny na vertexy, ale jsou složeny z úseček. Následně jsou seznamy prvků převedených na vertexy porovnávány s referenčními seznamy a to tímto způsobem:

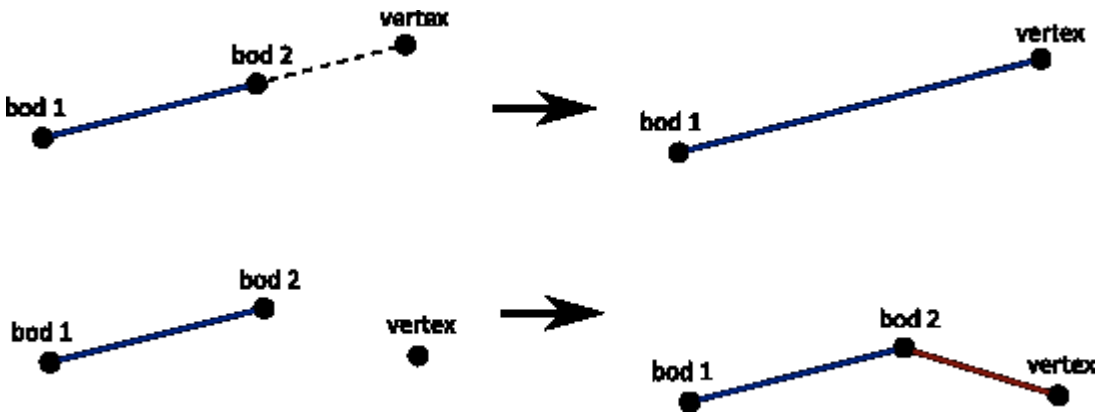
> Vezme se jeden vertex a najde se vzdálenost ke všem úsečkám z referenčního seznamu. Ponechá se jen ta nejmenší vzdálenost. Tohle se udělá pro všechny vertexy. Následně se sečtou kvadráty těchto vzdáleností a normují se na počet vertexů. To znamená, že je součet kvadrátů těchto vzdáleností podělen počtem vertexů znaku. A nakonec se seznam identifikuje jako znak, který měl

s příslušným referenčním seznamem nejmenší normovaný kvadrát těchto vzdáleností. A vše se opakuje pro všechny seznamy.

3.6.2 Implementace algoritmu pro vektorizaci obrazu

Na začátku jsou vytvořeny referenční seznamy s jednotlivými prvky, které jsou detekovány. Jako referenční prvky jsou použity čísla 0-9. Referenční seznam je vytvořen pomocí následujícího algoritmu.

Nejprve se vezme první bod úsečky (vertex) a hledá se další nejbližší vertex (druhý bod úsečky). Znovu se hledá nejbližší vertex druhého bodu. Pokud druhý bod úsečky leží mezi prvním bodem úsečky a tímto vertexem, nahradí se druhý bod tímto vertexem. To znamená, že druhý bod ležel na úsečce a nenesl žádnou další informaci – spojení dvou navazujících úseček. A opět se pokračuje tím, že se znovu hledá nejbližší vertex druhého bodu.



Obr. 24. Postup při vytváření referenčních seznamů prvků.

Dále se založí nová úsečka a celý postup se opakuje.

V první řadě se porovnávají referenční seznamy s detekovanými symboly. Pro lepší porovnání jednotlivých symbolů je symbolům odebrán jejich offset tak, aby jejich horní levý roh začínal v nule. Tohle provádí tato metoda.

```
public void OdebratOffset(List<Point> symbol)
```

První je získán vertex, který je nejvýš a nejvíc vlevo. Je předpokládáno, že to bude první vertex.

```
int minX = symbol.First().X, minY = symbol.First().Y;
```

Následující smyčka sloužící k průchodu ostatními vertexy. Obsahuje dvě podmínky, které vertexy porovnávají s minimem.

Tímto způsobem je zjištěna pozice levého horního rohu vertexu. Tato pozice je odečtena od všech vertexů. A tím je celý symbol posunut do horního levého rohu. Následně je offset odebrán všem symbolům.

Následující metodou je vrácena normovaná vzdálenost bodů symbolu od přímek z referenčního znaku.

```
public double PorovnatsReferencnimListem(Primka[] referencniList, List<Point>
symbol)
```

Dále je vytvořena proměnná ret. Do této proměnné jsou ukládány všechny součty kvadrátů vzdáleností úseček, které se pak normují podle počtu vertexů.

```
double ret = 0
```

První je spočítána vzdálenost všech úseček z referenčního znaku vůči jejich nejbližšímu bodu symbolu. Tato vzdálenost je přidána do proměnné ret. Následně jsou spočteny vzdálenosti bodů symbolu vůči jejich nejbližším úsečkám referenčního listu. Výsledek je přičten k proměnné ret.

Pro větší přesnost se vydělí kvadráty vzdáleností počtem bodů.

```
ret /= (symbol.Count);
return ret;
```

Poté jsou porovnány všechny načtené symboly s referenčními znaky. A výsledky tohoto porovnání jsou vykresleny a vypsány. To se provádí touto metodou.

```
public void PorovnatSymboly(Image cilovyObrazek)
```

Prvně se projde všemi dostupnými symboly a začnou se porovnávat.

Po vykreslení přímek se zjišťuje, jestli se dvě přímky protly a kde se protly. Toto zajišťuje tato metoda.

```
private float VzdalenostOdPrimky(Point bodPrimky1, Point bodPrimky2, Point p)
```

Vzdálenost bodu od přímky je počítána pomocí tohoto vzorce.

$$v = \frac{\left|a \cdot XA + b \cdot YA + c\right|}{\sqrt{\left(a \cdot a + b \cdot b\right)}} \quad (6)$$

První je získán směrový vektor, který je hned převeden na normálový. A vypočte se a, b, c.

```
int normalX = -(bodPrimky2.Y - bodPrimky1.Y);
int normalY = (bodPrimky2.X - bodPrimky1.X);
int parameter = -(bodPrimky1.X * normalX) - (bodPrimky1.Y * normalY);
```

Následně je vytvořen seznam pro vertexy a seznam přímek symbolu.

```
private List<Point> vertexy = new List<Point>();
```

Dále je nastavena maximální vzdálenost mezi vertexy podobně jako u segmentace. Následně je vytvořena metoda pro vykreslení vektorizovaných symbolů.

```
public void VykreslitVektorizovaneSymboly(Image obrazek)
```

Poté se nastaví černá barva pro vykreslení jednotlivých symbolů. Vertexy jsou převedeny na přímky následující metodou.

```
public void VertexynaPrimky(List<Point> symbol)
```

První se najde nejbližší bod k prvnímu bodu symbolu. Dále je přidána přímka, od které se budou odvozovat další. Projde se všemi přímkami a jejich body a najdou se k nim nejbližší body.

Dále se zvolí indexy nejbližších bodů vůči startovnímu a konečnému bodu úsečky.

```
Point startNear = sym[0], endNear = sym[0];
int startMinIndex = 0, endMinIndex = 0;
float distanceMinA = 0;
float distanceMinB = 0;
```

Dále se první zjistí nejbližší bod ke startovnímu bodu a nejbližší bod ke konečnému bodu.

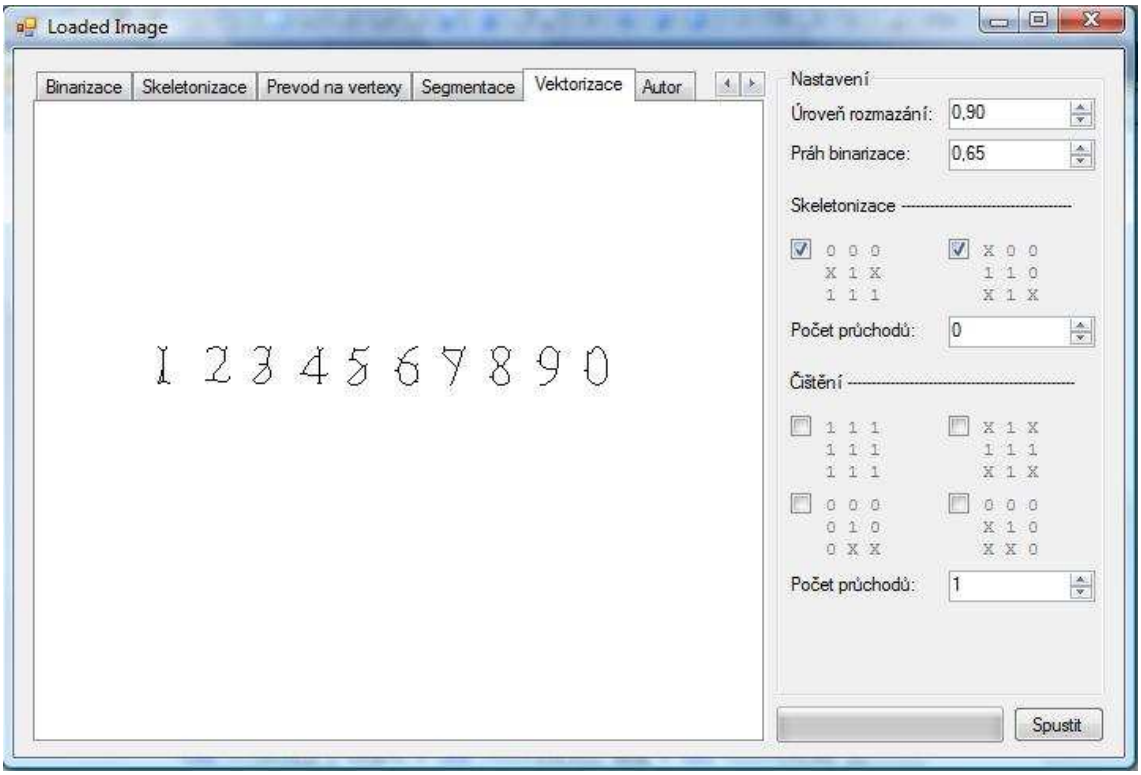
Poté jsou všechny symboly převedeny na přímky.

```
public void VsechnySymbolyNaPrimky(List<List<Point>> symboly)
```

Nakonec se barevně vykreslí symbol rozdělený na čáry. Respektive referenční symboly jsou vykresleny na načtené symboly. Takže v konečném důsledku vektorizované symboly zůstanou černé.

3.6.3 Výsledky implementace algoritmu pro vektorizaci obrazu

Obraz je vektorizován. Jednotlivé prvky v obrazu jsou porovnávány s referenčními znaky. Jako referenční znaky jsou zvoleny čísla s fontem Times New Roman.



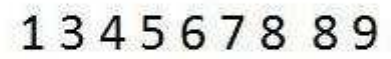
Obr. 25. Vektorizovaný obraz.

4. Testování systému

Testování systému lze provádět pomocí propojení vývojového kitu a PC přes sériové rozhraní. Toto testování se uplatní při vývoji algoritmu pro detekce geometrických prvků v obrazovém signálu. Veškeré výsledky jednotlivých úprav obrazového signálu jsou zobrazeny v jednotlivých záložkách uživatelské aplikace.

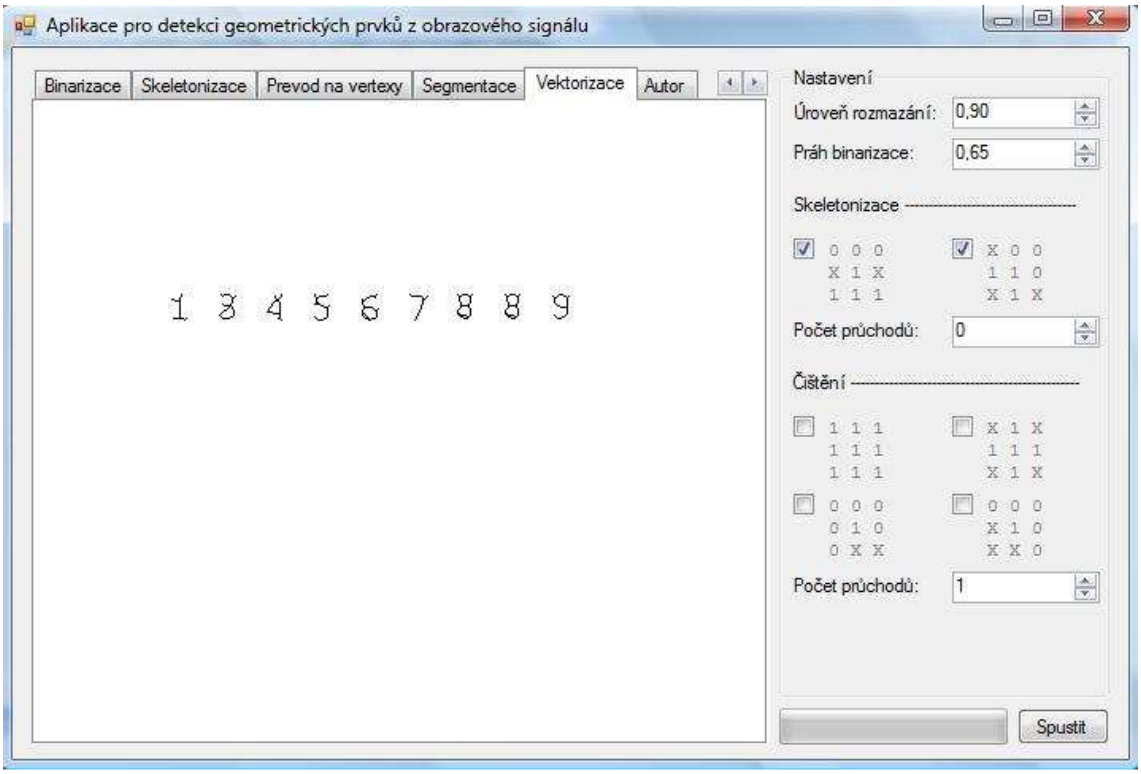
Obr. 26. Princip testování systému.

Pro testování algoritmu, který detekuje geometrické prvky z obrazového signálu jsou vybrány následující snímky.
První snímek obsahuje čísla 0-9. Velikost čísel je 20b. a zvolený font je Calibri.



Obr. 27. První testovaný snímek.

Po vektorizaci vypadá snímek následovně.



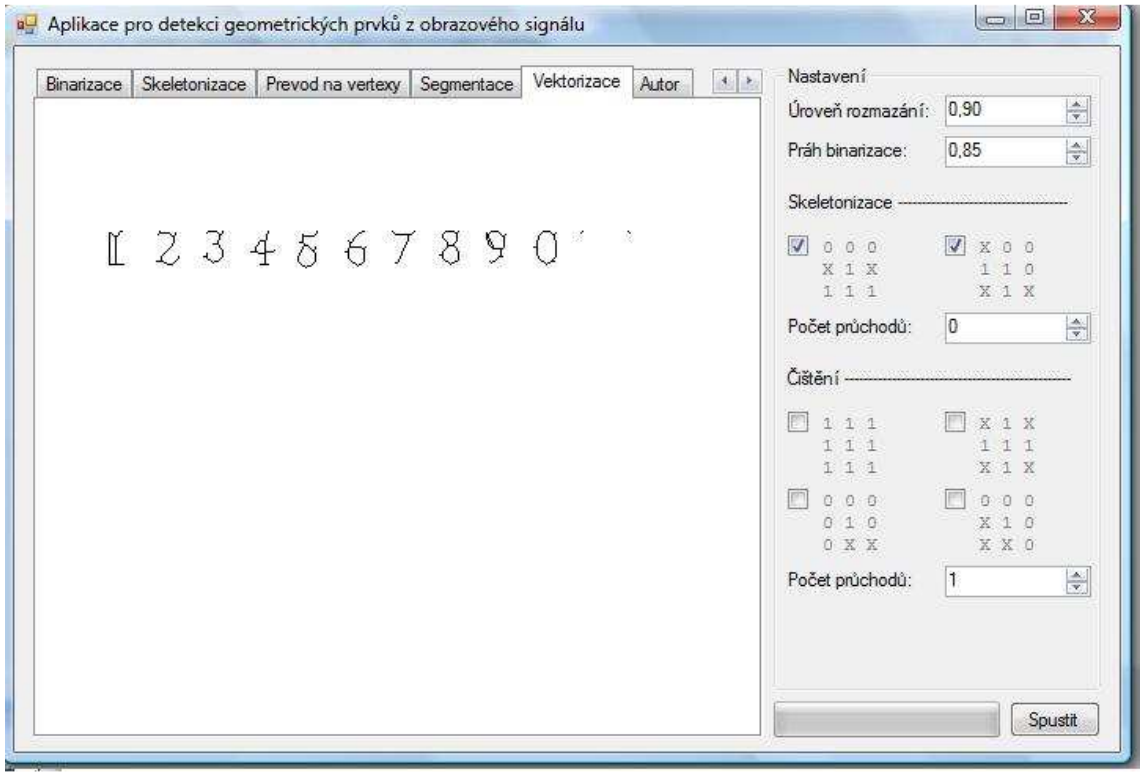
Obr. 28. Vektorizovaný snímek.

Je vidět, že vektorově detekovaná čísla obsahují různé přebytečné znaky. Některé tyto znaky jsou vytvořeny už při při skeletonizaci. Skeletonizace se provádí pomocí dvou hlavních jader. Tyto jádra jsou dány definicí. Existují čtyři další jádra ,které mohou být použity. Nicméně po jejich použití se buď obraz nezmění, nebo je ještě více poškozen. Pro vektorovou detekci byl nastaven práh binarizace na hodnotu 0,65 což je implicitně nastavená hodnota, která v tomto případě vyhovuje. Další nesrovnalosti vznikají při samotné vektorizaci, protože jako referenční znaky jsou použity čísla s jiným fontem, než mají prvky, které jsou detekovány.

Druhý snímek obsahuje modré čísla 0-9. Velikost čísel je 26b. a zvolený font je Centaur.

Obr. 29. Druhý testovaný snímek.

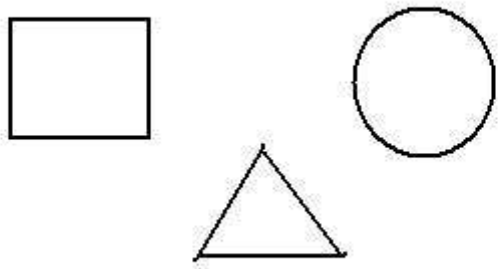
Po vektorizaci vypadá snímek následovně.



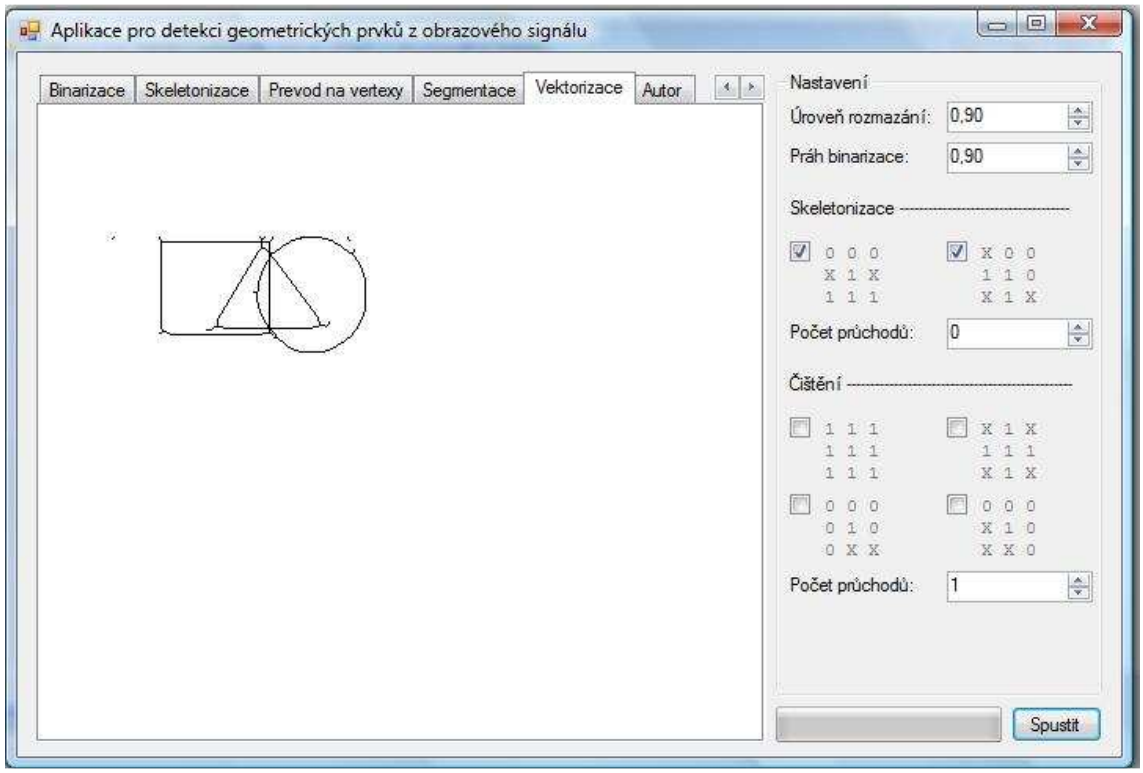
Obr. 30. Vektorizovaný snímek.

Jak je vidět tak i tady vznikají chyby způsobené jak skeletonizací tak i tím, že jsou jako referenční znaky použity čísla, která mají menší velikost a jiný font. Jelikož je vybrána modrá barva. Tak se práh binarizace musel zvýšit. Při původní přednastavené hodnotě byly jednotlivé prvky v obrazu skoro celé vymazány. Při postupném přidávání hodnoty prahu je zjištěno, že dostatečná hodnota je až 0,85. Při nižších hodnotách jsou prvky v obrazu natolik vymazány, že následná vektorizace není možná.

Následuje snímek s jednoduchými geometrickými prvky.



Obr. 30. Třetí testovaný snímek.



Obr. 32. Vektorizovaný snímek.

Také zde jsou pozorovány jisté chyby způsobené skeletonizací. Všechny detekované prvky se překrývají. To je způsobeno tím, že pro vektrovou detekci geometrických prvků je přednastavena pevná vzdálenost mezi jednotlivými prvky. Tato vzdálenost je určena pro čísla nikoliv pro jednoduché geometrické prvky této velikosti.

Výpočet práhu binarizace :

První práh T je určen náhodně. Obraz je rozdělen na pixely, které tvoří objekt a na pixely, které tvoří pozadí. Jsou vytvořeny dvě soustavy pixelů:

$$G_1 = \{\ f(m,n) : f(m,n) > T\ \} \quad (7) \text{ - Pixely objektu}$$

$$G_2 = \{\ f(m,n) : f(m,n) \le T\ \} \quad (8) \text{ - Pixely pozadí}$$

Hodnoty pixelů $f(m,n)$ jsou nalezené v m-tém sloupci a v n-tém řádku. Následně je spočítána průměrná hodnota jednotlivých soustav pixelů.

$$m_1 = \text{průměr z } G_1 \quad (9)$$

$$m_2 = \text{průměr z } G_2 \quad (10)$$

Nový práh je vytvořen výpočtem průměrné hodnoty z m_1, m_2.

$$T' = \frac{(m_1 + m_2)}{2} \quad (11)$$

Vektorizace

Segmentací byly asociovány body do jednotlivých seznamů, které reprezentují znaky. Nyní je potřeba každý znak tvořený z bodů převést na znak tvořený úsečkami.

Postup

1. První byla vytvořena referenční úsečka, ke které se budou následně připojovat další. Vezme se tedy první bod ze seznamu prvního znaku a najdeme k němu nejbližší bod. Z těchto dvou bodů sestrojíme referenční úsečku.
2. Dalším krokem je iterace příslušného seznamu vertexů a hledání nejbližších bodů k bodům vymezujícím začátek a konec přímky. Po nalezení takových bodů se porovná, který z nich je blíž svému referenčnímu bodu (buď k bodu začátečnímu, nebo bodu na konci přímky) a utvoří se nová přímka z těchto dvou bodů.

Obr. 33. Vektorizace.

3. Bod dvě se provede pro všechny dosud přidané přímky.
4. Vyčerpají-li se všechny body znaku, vektorizace příslušného znaku končí.
5. Pokračuje se dalším znakem bodem jedna.

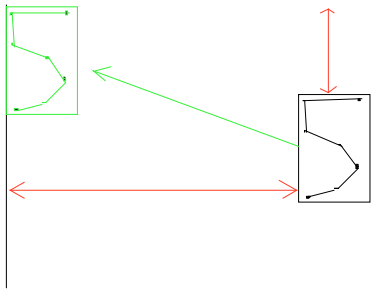
Odebrání offsetu a vykreslení znaků

Každý znak v obraze má svou specifickou pozici - offset. Teneto offset je promítnutý do každého jednotlivého vertexu ve znaku. Pro plnou kontrolu nad tím, kde se znak vypisuje a pro ušetření práce při vzájemném porovnávání s referenčními znaky, které offset nemají, je offset od znaku odstraněn. A znak je doslova přesunut na počátek souřadné soustavy.

Postup

1. Zpočátku je zjištěno, jak velký offset vlastně je. Prakticky jde o pozici levého horního vertexu znaku. Tento vertex musí být nalezen. V seznamu se tedy najde vertex s nejmenší Xovou a Ynouvou souřadnicí. Takové vertexy mohou být i dva (Vertex s nejmenší Xovou souřadnicí a další s nejmenší Yovou souřadnicí). Nicméně v tuto chvíli se pozornost věnuje nejmenším souřadnicovým hodnotám a ne konkrétním vertexům.

2. Po té, co je offset nalezen, tak je odečten od všech vertexů nacházejících se v příslušném znaku, a tím je znak přesunut na počátek souřadné soustavy.



Obr. 34. Odebrání offsetu a vykreslení znaků.

3. Výše uvedený postup je aplikován na všechny znaky.
4. Znaky pak mohou být vykresleny na libovolné pozici.

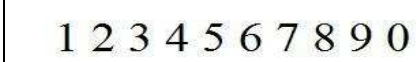
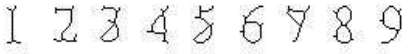
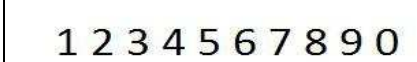
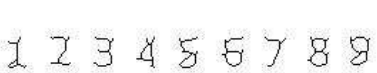
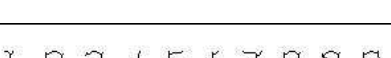
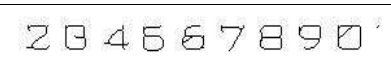
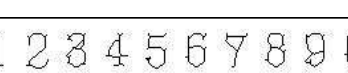
Porovnávání s referenčními symboly

V aplikaci jsou uloženy seznamy úseček reprezentující jednotlivé symboly, které slouží jako vzor pro porovnávání a asociaci se vstupními symboly.

Postup při porovnávání

1. Nejprve je vybrán první symbol tvořený body (Pro zvolený algoritmus porovnání není potřeba převádět načtené body symbolů na úsečky. Znaky postačí ve formátu bodů)
2. Projde se seznamem bodů a je zjištěno, kolik bodů vstupního znaku leží na některé z přímek referenčního znaku. Tímto způsobem je zjištěno kolik bodů leží mimo přímky referenčního seznamu.
3. Následně se provede opačný proces. Projde se jednotlivými přímkami referenčního znaku a zjišťuje se, jestli na nich neleží některý z bodů vstupního znaku. Tímto způsobem je zjištěno kolik přímek referenčního znaku není použito.
4. Procentuální přesnost je získána vydělením počtu shod celkovým počtem přímek a bodů obou znaků (vstupního a referenčního) a následným vynásobením sty.
5. Znaku se přiřadí jeho referenční znak dle nejvyšší shody.
6. Tímto způsobem se zkontrolují všechny načtené znaky.

Systém pro analýzu a detekci geometrických prvků byl v první řadě otestován na číslech. Jednotlivé řady čísel mají různé fonty. Testované snímky 6-10 mají kromě jiných fontů také různé barvy.

Číslo snímku	Původní snímek	Vektorizovaný snímek
1.		
2.		
3.		
4.		
5.		
6.		
7.		
8.		
9.		
10.		

Tab. 2. Původní a vektorizované snímky.

Číslo snímku	Přesnost vektorizace [%]										Práh binarizace
	Pořadí znaků										
	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	
1.	97,67	70,27	65,15	86,76	71,43	84,88	59,32	82,56	87,18	-	0,70
2.	54,24	56,25	66,67	57,14	70	58,97	61,82	58,33	68,42	-	0,70

3.	28,95	67,57	84,71	86,76	85,37	68,12	48,33	80,65	68,57	80,22	0,65
4.	77,27	71,26	80,21	78,75	81,37	84,91	67,61	80,56	77,88	0	0,80
5.	47,27	74,32	64,47	81,08	91,11	69,77	61,67	77,89	64,71	81,32	0,65
6.	24,14	60,81	58,67	69,74	58,67	75	54,24	76,77	56,41	52,44	0,70
7.	62,30	78,89	84,21	65,43	97,85	91,92	83,33	89	67,78	85,42	0,80
8.	61,54	70,91	29,41	52,73	49,02	47,92	62,75	40,35	20	29,63	0,55
9.	41,86	51,16	64,10	53,85	82,83	61,63	0	46,77	0	76,04	0,70
10.	0	37,21	85,42	71,25	79,55	78,57	75,53	66,67	0	16,67	0,90

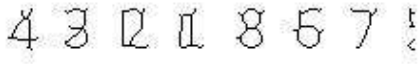
Tab. 3. Přesnost vektorizace jednotlivých znaků vyjádřena v procentech.

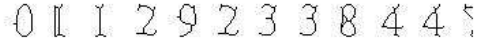
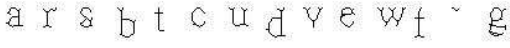
U prvních dvou snímků je vidět, že znak, který reprezentuje číslo nula není vektorizován. To je způsobeno tím, že při binarizaci vznikají nad nulami nadbytečné znaky. Díky těmto znakům (čárám o velikosti jen několika pixelů) je číslo nula při vektorizaci úplně vypuštěno. Na snímku číslo šest je vidět, že jsou vektorizovány všechny znaky. Při binarizaci vznikly sice další dva znaky, které systém registruje, ale ty jsou zanedbatelné. Při testování snímku osm musel být práh binarizace snížen až na 0,55, protože zvolený font má velkou šířku. V konečném výsledku se ukázalo, že tento font není vektorizován s velkou přesností. Většina znaků je naprosto nerozeznatelná. Na snímku číslo devět je ukázáno, že díky zvolenému fontu se od čísla sedm oddělil vizuálně takřka neviditelný znak. Díky tomuto znaku je číslo sedm posunuto a číslo nula je umazáno. Kvůli nepatrnému výčnělku na čísle osm je toto číslo identifikováno jako jednička a proto je přesnost uvedená v tabulce 3. nula procent. Při vektorizaci snímku číslo deset byl práh binarizace zvolen na hodnotu 0,90, protože zvolená barva je příliš světlá a při původní přednastavené hodnotě je obraz natolik umazán, že vektorizace není možná.

Procentuální přesnost jednotlivých znaků uvedená v tabulce 3., se vztahuje k referenčním znakům.

Jako referenční znaky jsou použity čísla 0-9 fontu Times New Roman velikosti 26b.

V další částí testování systému pro analýzu a detekci geometrických prvků se testují snímky s víceřádkovým textem a číslicemi.

Číslo snímku	Původní snímek	Vektorizovaný snímek
1.	159370 260481 371595 482600	

2.	1 2 3 4 5 6 7 8 9 0 0 9 8 7 6 5 4 3 2 1 1 2 3 4 5 6 7 8 9 0	
3.	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	
4.	a b c d e f g h i j k l m n o p q r s t u v w x y z	

Tab. 4. Původní a vektorizované snímky.

Při testování více řádkového uspořádání číslic je zjištěno, že systém není na tuto situaci dimenzován. Během vektorizace víceřádkového uspořádání zobrazí jen některé znaky. A tyto znaky jsou vykresleny v náhodném pořadí.

Jak je ze snímku číslo tři a čtyři vidět systém si také poradí s vektorizací písmen.

5. Závěr

Cílem této bakalářské práce bylo navrhnout a realizovat systém pro analýzu a detekci gemotetrických prvků z obrazového signálu.

Řešení je rozděleno do dílčích úprav obrazového signálu. V první řadě se provádí Gaussovo rozmazání, které odstraňuje obrazové šumy což mohou být různé nečistoty jako šmouhy, různé čáry, kazy.

V dalším kroku se obrázek binarizuje tzn., že se převede na černobílý podle práhu jasu. Pokud obrázek není ve formátu s barvami RGB musí se převést. Dále se získá jas pixelů a to tak že se udělá průměr z RGB. Pokud je jas menší než práh pak se pixel nastaví na černou barvu (jinak se nastaví na bílou).

Poté následuje metoda skeletonizace. Obraz je ztenčen metodou popsanou výše. Tuto metu provádíme proto, abychom ztenčili obrysy prvků v obrazu na šířku jednoho pixelu. Po této úpravě se nám obrázek lépe vektorizoval.

Obraz byl převeden na vertexy. Vertexy jsou reprezentovány jako souřadnice bodů, ze kterých se prvek skládá. Dále byla provedena tzv. segmentace obrazu, kde byly jednotlivé prvky barevně odděleny a převedeny z vertexů zpět na jednotlivé symboly.

Ještě před samotnou vektorizací byly prvky, které mají být detekovány převedeny na úsečky a uloženy do jednotlivých referenčních seznamů.

Následná vektorizace se provádí tímto způsobem. Obraz byl opět převeden na vertexy. Poté se vzal jeden vertex a našla se vzdálenost ke všem úsečkám z referenčního seznamu. Ponechala se ta

nejmenší vzdálenost. Tohle se udělalo pro všechny vertexy. Následně byly sečteny kvadráty těchto vzdáleností a kvadráty byly normalizovány na počet vertexů. To znamená, že součet kvadrátů těchto vzdáleností byl podělen počtem vertexů znaku. Nakonec byl seznam vertexů identifikován jako znak, který mel s příslušným referenčním znakem nejmenší normovaný kvadrát těchto vzdáleností. Vše se opakovalo pro všechny seznamy a prvky byly vektorizovány. A to tak že referenční znak překreslil detekovaný znak.

Veškeré dílčí úpravy obrazového signálu jsou zobrazeny v záložkách uživatelské aplikace.

V celkovém výsledku vektorová detekce geometrických prvků za použití referenčních seznamů není příliš přesná. Tato detekce může být použita jen pro určité tvary, které jsou použity jako referenční. Chyby vznikají už při malých rozdílech s těmito referenčními znaky. Čím je více znaků tím déle trvají výpočty jednotlivých úprav obrazového signálu. Zrychlení systému by se dalo dosáhnout jak návrhem úspornějšího algoritmu, tak i volbou jiného programovacího jazyka a to např. jazyka C, C++.

V této práci jsem se naučil používat metody zpracování a úpravu obrazového signálu, které jsem dříve neznal. Naprogramování těchto metod bylo pro mě jak časově tak myšlenkově náročné. Musel jsem si značně prohloubit své znalosti v programovacím jazyku C#.

Program byl vytvořen v programovacím prostředí Microsoft Visual Studio 2010 v jazyce C#.

Literatura:

[1] BEZDĚK, Miroslav. Elektronika III. České Budějovice: Kopp, 2004. 236 s. ISBN 80-7232-241-9.

[2] BURKHARD, Kainka. USB:měření, řízení a regulace pomocí sběrnice USB. Praha: BEN-technická literatura, 2002. 256 s. ISBN 80-7300-073-3.

[3] VÍT, Vladimír; GREGORA, Pavel. Televizní technika:Zařízení pro přenos

a vysílání televizního signálu. Praha: BEN-Technická literatura, 2000. 176 s. ISBN 80-86056-89-9.

[4] VÍT, Vladimír. Televizní technika: přenosové barevné soustavy. 1. vyd. Praha: BEN-technická literatura, 1997. 720 s. ISBN 80-86056-04-X.

[5] VÍT, Vladimír; PETR, Kuba. Televizní technika: Studiové zpracování televizního signálu. 1. vyd. Praha: BEN-technická literatura, 2000. 208 s. ISBN 80-86056-88-0.

[6] *Wikipedie* [online]. 2010-01-18 [cit. 2011-01-19]. RGB. Dostupné z WWW: <http://cs.wikipedia.org/wiki/RGB>

[7] [8] ŠTRBÍK, Ondřej. Vestavěný řídicí systém pro detekci trajektorie pohybujícího se zařízení. [s.l.], 2010. 37 s. Bakalářská práce. VŠB – Technická univerzita Ostrava.

[9] Gaussian blur - Wikipedia, the free encyclopedia [online]. 2011-04-06 [cit. 2011-04-28]. Gaussian blur. Dostupné z WWW: <http://en.wikipedia.org/wiki/Gaussian_blur>.

[10] HRACH, Rudolf. Obrazová informace v počítačové fyzice [online]. 1997-10-26 [cit. 2011-04-28]. Obrazová informace v počítačové fyzice. Dostupné z WWW: <http://www.kolej.mff.cuni.cz/~lmotm275/RUZE/07/b/node1.html>.

[11] FISHER, Robert, et al. Morphology - Thinning [online]. 2003 [cit. 2011-04-28]. Thinning. Dostupné z WWW: <http://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm>.

[12] PALÁGYI, Kálmán. Skeletonization [online]. 2000 [cit. 2011-05-01]. Skeletonization. Dostupné z WWW: <http://www.inf.u-szeged.hu/~palagyi/skel/skel.html>.

[13] Segmentation (image processing) - Wikipedia, the free encyclopedia [online]. 25.4.2011 [cit. 2011-05-05]. Segmentation (image processing). Dostupné z WWW: <http://en.wikipedia.org/wiki/Segmentation_(image_processing)>.

[14] Vertex (computer graphics) - Wikipedia, the free encyclopedia [online]. 24.8.2010 [cit. 2011-05-05]. Vertex (computer graphics). Dostupné z WWW: <http://en.wikipedia.org/wiki/Vertex_(computer_graphics)>.

[15] MCGUIGAN, Brendan. What is a Vectorized Image? [online]. 5.3.2011 [cit. 2011-05-08]. Hat is a Vectorized Image?. Dostupné z WWW: <http://www.wisegeek.com/what-is-a-vectorized-image.htm>.

Seznam příloh:

Příloha I Zdrojový kód Detekce geometrický tvarů pomocí barvy vytvořený v Matlabu

Příloha II Zdrojový kód algoritmu Gaussovského rozmazání vytvořený v Microsoft Visual Studiu 2010

Příloha III Zdrojový kód algoritmu Binarizace obrazu vytvořený v Microsoft Visual Studiu 2010

Příloha IV Zdrojový kód algoritmu Skeletonizace obrazu vytvořený v Microsoft Visual Studiu 2010

Příloha VI Zdrojový kód algoritmu Segmentace vytvořený v Microsoft Visual Studiu 2010

Příloha VII Zdrojový kód algoritmu Vektorizace vytvořený v Microsoft Visual Studiu 2010

Příloha VIII Datasheet k vývojovému kitu CMUcam3 (CMUcam3_datasheet.pdf)

Veškeré přílohy jsou dodány v elektronické podobě na CD